lets you flex, bend and deform the character as you like through interactive Bendy Bones controls.

Shareable animation through all Rigify rigs

Since the control system is generated by Rigify, if you share a meta-rig through different characters you will be able to share data between them even if they have different proportions.

Extendable feature set

You can save and encode your meta-rigs to a button to have them available at any time without recreating it by hand or share your meta-rigs with other people. Through Python scripting you can also extend Rigify with new Rigify-types or new rig samples by implementing your own feature set package.

Ready to go

Once you generate your rig you won't need Rigify or any other add-on to use it.

Enabling Add-on

1. Open Blender and go to Add-ons section of the Preferences.
2. Search "Rigify" and check the *Enable Extension* checkbox.

Basic Usage

Basic Rig Generation

1. Add a meta-rig structure from the *Add ▸ Armature* menu.
2. Edit the bone positions to match the character geometry.
3. In the armature properties click on the *Generate Rig* button to generate the rig.

Add a Predefined Meta-Rig

Reference

Mode::	Object Mode
Menu::	*Add ▸ Armature*
Shortcut::	`Shift-A`

Rigify stores all the information required to generate complex rig controls and mechanism in more simple armatures called “meta-rigs”.

The predefined meta-rigs can be found in the *Add* menu. Currently available meta-rig types are:

- Basic Human (doesn’t include face and fingers)
- Basic Quadruped
- Human
- Cat
- Wolf
- Horse
- Shark

Edit Bone Positions

To correctly match your character, meta-rig bones must be moved to correct positions. This can be achieved in two different ways: Pose Mode or Edit Mode.

Note

Rigify assumes that 1 unit corresponds to 1 meter. So a human is about 2 units tall. If your character is in a different scale and you are more familiar with modeling rather than rigging, it is suggested to scale it to Rigify dimensions before positioning the meta-rig bones. If you want to scale the character's geometry, we suggest you to first scale up the character in Object Mode, then apply the geometry scale with the *Apply Scale* tool.

Rigify Human Alignment Tips

- Limbs: Keep the legs as straight as possible in the front view (Rigify human works better in predictable cases). Give the knee and the elbow a slight bend angle (Rigify needs to know where your knee/elbow is pointing).

- Torso: Keep the spine as straight as possible in the front view (Rigify human works better in predictable cases). The last bone of the spine is the head. By default the next two bones (top to bottom) are considered the neck bones. It is suggested to keep the neck bones as aligned as possible while editing.

- Face: Positioning face bones can be tricky if you are not an expert in bone editing and they are almost useless if you plan to make facial animation through shape keys. Consider removing face features from your character if they aren't really needed. If you don't need the face all the face bones can be deleted. All the face bones are in the *Face* armature bone collection by default. You can select them by displaying only that collection, selecting all of its content and then deleting the bones in Edit Mode to correctly remove the face.

If you want to scale all the face bones at once, consider scaling the face master bone in Pose Mode (see Pose Mode matching method). The face master bone is placed in the same position of the head bone. To select it easily, hide all other bone collections.

For more tips, see the Positioning Guide.

Pose Mode Matching (Basic)

Enter the meta-rig Pose Mode. Rotate, scale, and translate the bones in the correct position. When bones are in correct positions (always staying in Pose Mode) use *Apply* ▸ *Apply Pose As Rest Pose*.

Note

Connected bones cannot be translated in Pose Mode. You can scale the parent bones to match the general length and then refine child bones scale. For more detailed information on armature modes please refer to the armatures section.

Edit Mode Matching (Advanced)

Some basic armature display setup is suggested before entering bone Edit Mode.

With the meta-rig selected, go in the Properties and click on the Object tab. Scroll down to the display panel and enable X-ray and under *Maximum Draw Type* selector select *Wire*. This way the bones will always be drawn in wireframe on top of your geometry.

Then, always in the Properties click on the Armatures tab and under display check the *Axis* checkbox. This way you the bones rotation axes will be displayed during the edit process.

For more detailed information on armature display modes please refer to the Display panel page.

Generating the Rig

With the bones in the correct positions, jump back in Object Mode, go to the Armature tab, scroll down to the bottom and click on the *Generate Rig* button to finalize the rig creation. The generation process will take from few seconds to one minute depending on rig complexity and hardware specifications of your machine. If the generated rig needs tweaking, you can modify the meta-rig accordingly and then click again on the generate button. If the rig already exists, Rigify will simply overwrite it retaining all your modifiers and constraints and – where possible – all the previously generated features.

For information about additional generation options, see the Advanced Rig Generation section.

Tip

If the metarig uses the legacy face rig, you can use the *Upgrade Face Rig* button that appears above *Generate Rig* to automatically upgrade to the new modular face system.

The upgrade will preserve compatibility with existing skinning, but existing poses and animations will likely not be compatible due to subtle changes in control behavior.

Note

To make the rig overwriting work as expected, you need to have **both** the rig and the meta-rig visible before generating again. Rigify will try to unhide them in simple cases, but will abort generation if that fails.

Warning

As with all Python add-ons, Blender interface cannot be updated until the Python script execution is over. Wait until the rig appears to see the results.

Warning

Rigify is designed assuming a workflow where the meta-rig is kept available to allow re-generating the main rig whenever it is necessary to make changes to it. Removing the meta-rig after generating the main rig, or significantly modifying the generated rig is not advised: it will make it impossible to introduce features added in later versions of Rigify, or easily adapt it to breaking changes in later Blender versions. In general, automatic version update scripts will be provided for meta-rigs when necessary, but not generated rigs.

Binding the Geometry to the Rig

To bind the geometry to the rig you can use your preferred tools. Just few things you have to know:

- All the deforming bones are in the *DEF* bone collection.
- Eyes and Teeth bones of the legacy face are not deforming. You are supposed to bind the eyes and teeth geometry through Child Of constraints.
- Usually armature deform with automatic weights do a really good job out of the box if you correctly place your bones (and there is enough topology to work with!).

For more detailed information on bone collections, Armature modifier and weight painting refer to the Blender manual.

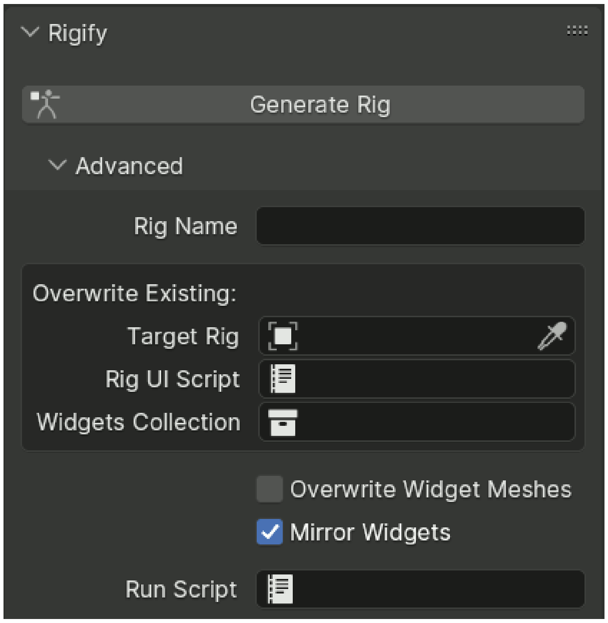
Advanced Rig Generation

Advanced Options Features

By using options in the Advanced sub-panel, it is possible to:

- Generate more than one rig per scene.
- Update/Override a specific rig.
- Force previously generated widget objects to be overwritten.
- Choose whether to use linked duplicates for left and right side widgets.
- Execute a script data-block after generation.

Advanced Options Sub-Panel



Rigify

Generate Rig

Advanced

Rig Name

Overwrite Existing:

Target Rig

Rig UI Script

Widgets Collection

☐ Overwrite Widget Meshes

☑ Mirror Widgets

Run Script

Advanced rig generation options are by default hidden in a sub-panel. Click on the *Advanced* line to open it.

Some of the options will be automatically set by Rigify if they have no value when a rig is generated, while others are fully controlled by the user.

Rig Name

When a brand new rig is generated, as opposed to overwriting an existing one, the value of this option is used to name it.

If this field is empty, the new object will be named based on the name of the metarig according to the following rules:

- If the name contains `META`, it is replaced with `RIG`.
- If the name contains `metarig`, it is replaced with `rig`.
- Otherwise, `RIG-` is prepended to the name.

When overwriting an existing rig object specified by the *Target Rig* option, its name is not changed, allowing it to be freely renamed without having to keep the value of this option in sync.

Target Rig *auto*

This option specifies the generated rig to overwrite when re-generating from this metarig.

If the option is not set, Rigify will generate a new rig object and store it in this option.

Note

When the option isn't set, Rigify will create a brand new rig object even if an object with a matching name already exists.

Rig UI Script *auto*

This option specifies the generated script datablock to overwrite when re-generating, and works in the same manner as *Target Rig*.

The script controls the UI in the 3D Viewport that allows conveniently switching visible bone collections, changing custom properties, converting between IK and FK and so on.

Widgets Collection *auto*

This reference specifies the collection containing generated widgets, and works in the same manner as *Target Rig*.

Overwrite Widget Meshes

If enabled, Rigify will generate new widgets every time the rig is re-generated. By default, it tries to reuse the already generated widget objects that exist in the widget collection, allowing them to be manually edited to fit the character better.

Mirror Widgets

When enabled, Rigify generates widgets for left and right side bones as linked duplicates, using negative X scale to flip the right side version. This enforces symmetry and reduces the number of meshes to adjust to fit the character.

When reusing an already generated widget, Rigify detects if it was originally generated mirrored by checking object scale to avoid flipping existing controls. Therefore switching to mirrored widgets for an existing character requires deleting the right side widgets, or *Force Widget Update*.

Run Script

It is possible to configure Rigify to execute a Python script contained in a text data-block after generation in order to apply user-defined customizations. The script is executed with the generated rig active and selected in Object Mode.

The simplest use of this may be adjusting properties of generated constraints when Rigify rig types don't have any relevant meta-rig settings. That can be done by using the *Copy Full Data Path* context menu option on the property, pasting it into the script and making an assignment, e.g.:

```
import bpy

bpy.data.objects["rig"].pose.bones["MCH-
spine.003"].constraints[0].influence = 0.6
```

Doing such changes via a script ensures they aren't lost if the rig is re-generated.

Users familiar with Rigify scripting can import Rigify utility modules, and access the generator instance through `rigify.get_generator()`. Yet note that, since generation is already finished, the only use of that is reading data created in the generation process.

Library Linking

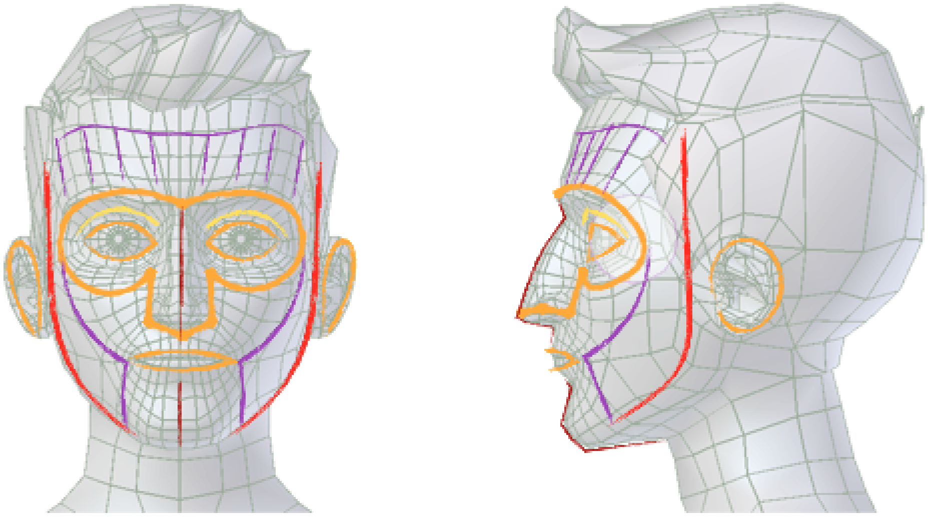
When linking a rig into another file, you generally want to create a collection that includes the generated rig and the character mesh. You do not need to include the meta-rig or the widget object collection. You then link in the collection and run Make Library Override.

The `rig_ui_template.py` text data-block responsible for the rig UI will be automatically linked along with the rig, you don’t need to link it separately. However, the script will not run until you run it manually from the Text editor or save and restart Blender.

Bone Positioning Guide

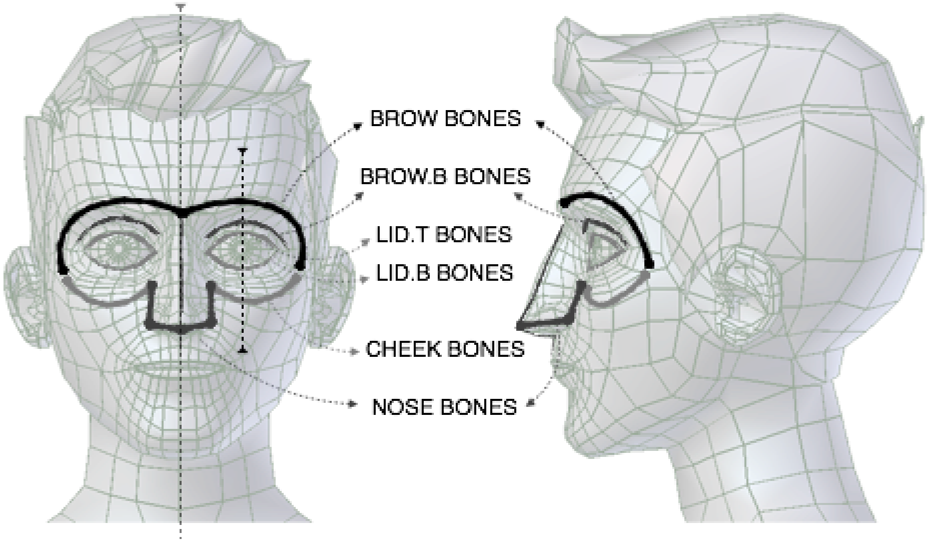
Face Bones

Start by identifying basic face landmarks to follow as guide for bones placement.



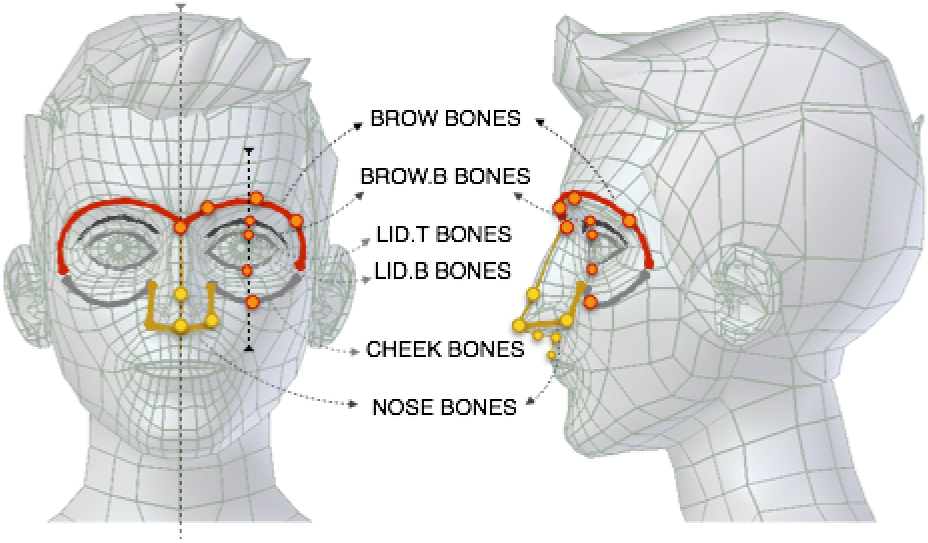
Basic Face Landmarks.

- Orange lines represent bones that should be placed in closed loops.
- Yellow lines represent bones whose position depends on surrounding bone loops.
- Red lines represent outer edge bones.
- Purple lines represent bridging bones used to cover deforming flesh.


BROW BONES
BROW.B BONES
LID.T BONES
LID.B BONES
CHEEK BONES
NOSE BONES

Eyes-Nose Landmarks.

The eyes-nose loop area is split in different parts identified by bone names. Follow the image to place the bones.

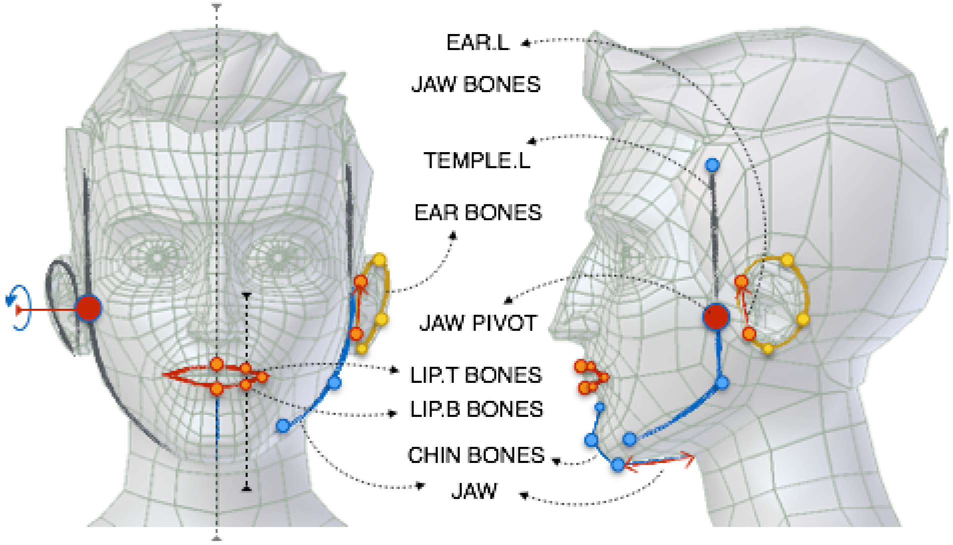

BROW BONES
BROW.B BONES
LID.T BONES
LID.B BONES
CHEEK BONES
NOSE BONES

Eyes-Nose Bone Positions.

Tip

Brow Placement

Keeping aligned the mid bones in “brow”, “brow.b”, “lid.t”, “lid.t” and cheek will give better results after rig generation.


EAR.L
JAW BONES
TEMPLE.L
EAR BONES
JAW PIVOT
LIP.T BONES
LIP.B BONES
CHIN BONES
JAW

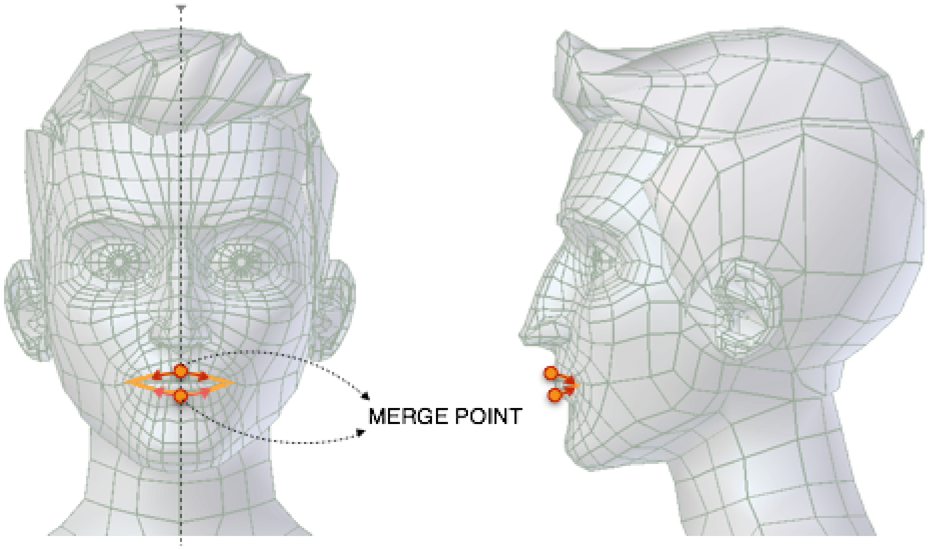
Jaw-Ear Bone Positions.

Also the jaw-ear area is split in different parts identified by bone names. Follow the image to place the bones.

Tip

Jaw Placement

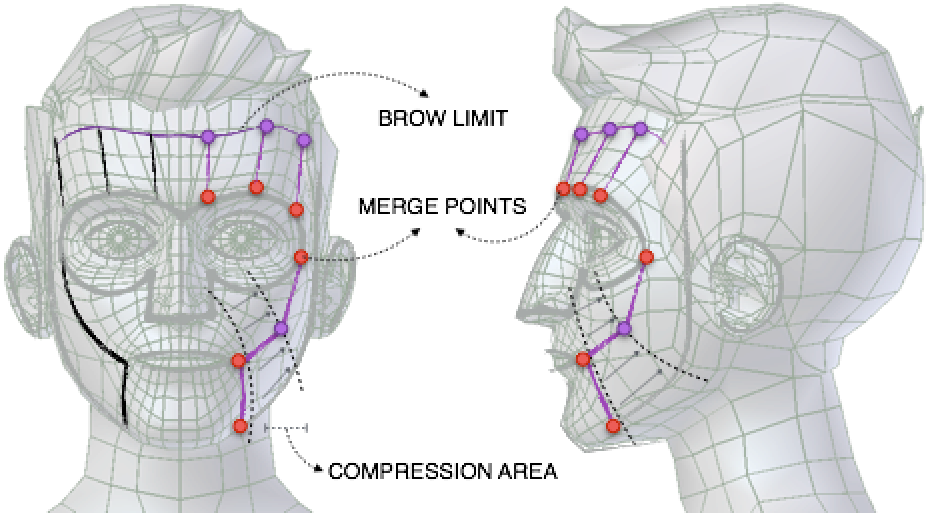
Try to place “ear.L” bone covering the part of the ear attached to the mandible (lower jaw). Do the same with temple bone trying to cover the part you don’t want to move with the jaw, this way you will also determine the jaw pivot position.


MERGE POINT

Lips Merge Point.

Warning

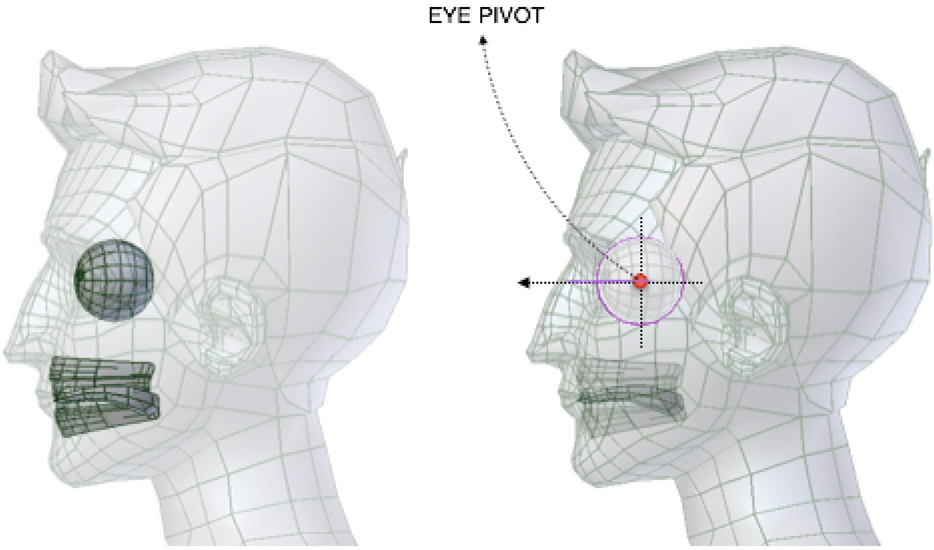
While moving the face bones it is necessary to preserve merge points, i.e. whenever heads or tails of two or more bones overlap at the same point, they should still do so after repositioning. Tearing a merge point apart may result in multiple controls being created instead of one, or even the generation of errors.


BROW LIMIT
MERGE POINTS
COMPRESSION AREA

Face Stretcher Bones.

After the main face bones are placed use the cheek bone to connect the eye-nose area to the jaw mouth area. Then do the same with the brow area. This process will automatically define face muscles compression areas.

Position the eye bones in the eye pivot point facing right **toward** the face on the Y axis. The length of the eye bones should correspond to the radius of the eye.



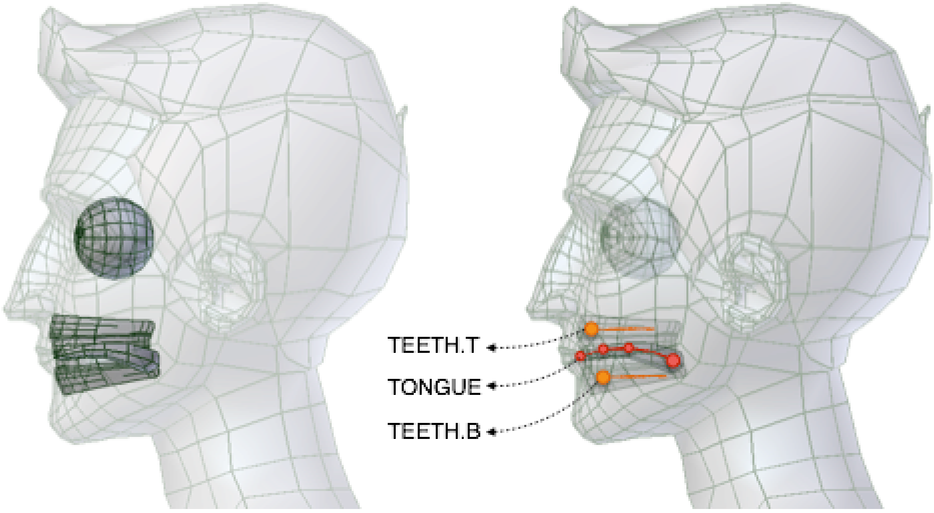
Eyes Pivot Position.

Tip

Eye Pivot

If your eye has a spherical shape you can define its pivot by entering Edit Mode. Select two opposite vertices on the center meridian – or the opposite poles – and snapping the cursor to selection by pressing *Snap ‣ Cursor To Selected*. If your eye is a complete sphere and its location it's not applied, then you can just use its center of mass.

Finally position the teeth bones on your teeth geometry and the tongue bone chain as described in the figure.



Mouth and Teeth Positions.

Tip

Tongue

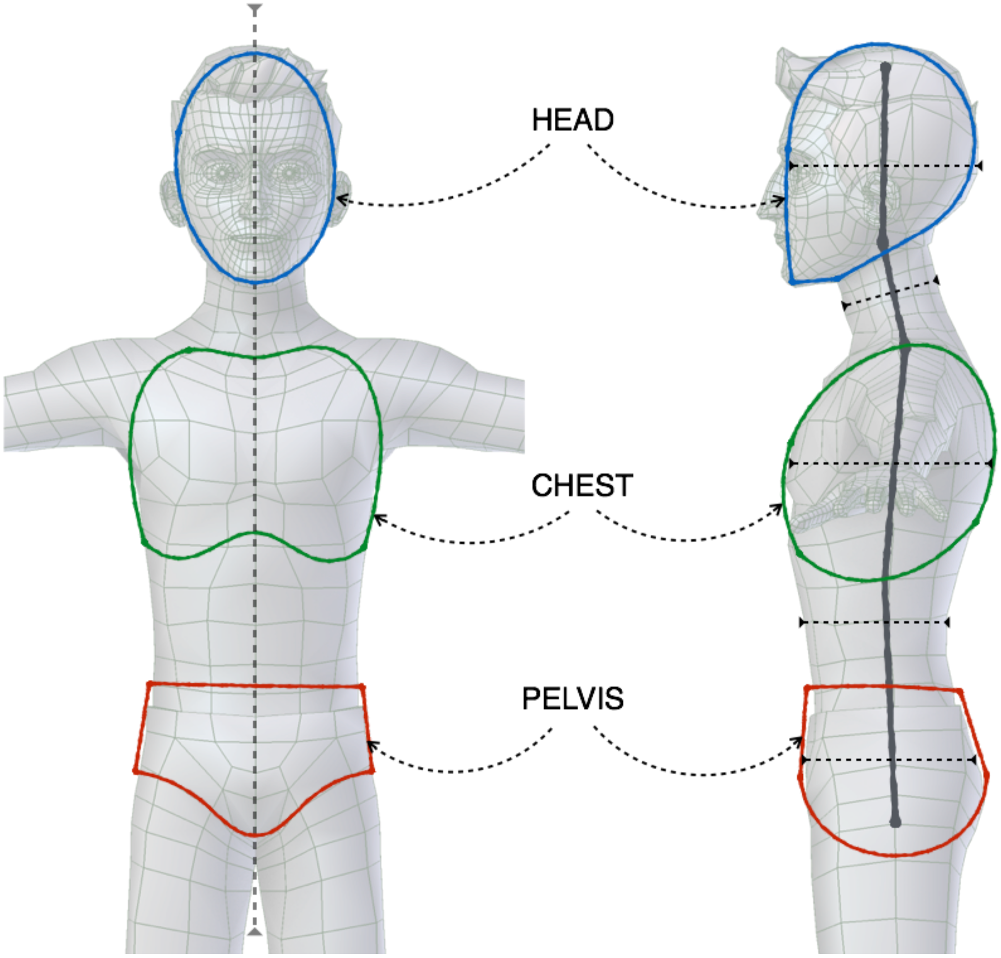
The tongue will work better if the bones are aligned at the symmetry line.

Before generating the rig ensure the face master bone is facing upward.

Torso Bones

Start by identifying on your character basic torso zones to follow as guide for bones placement.

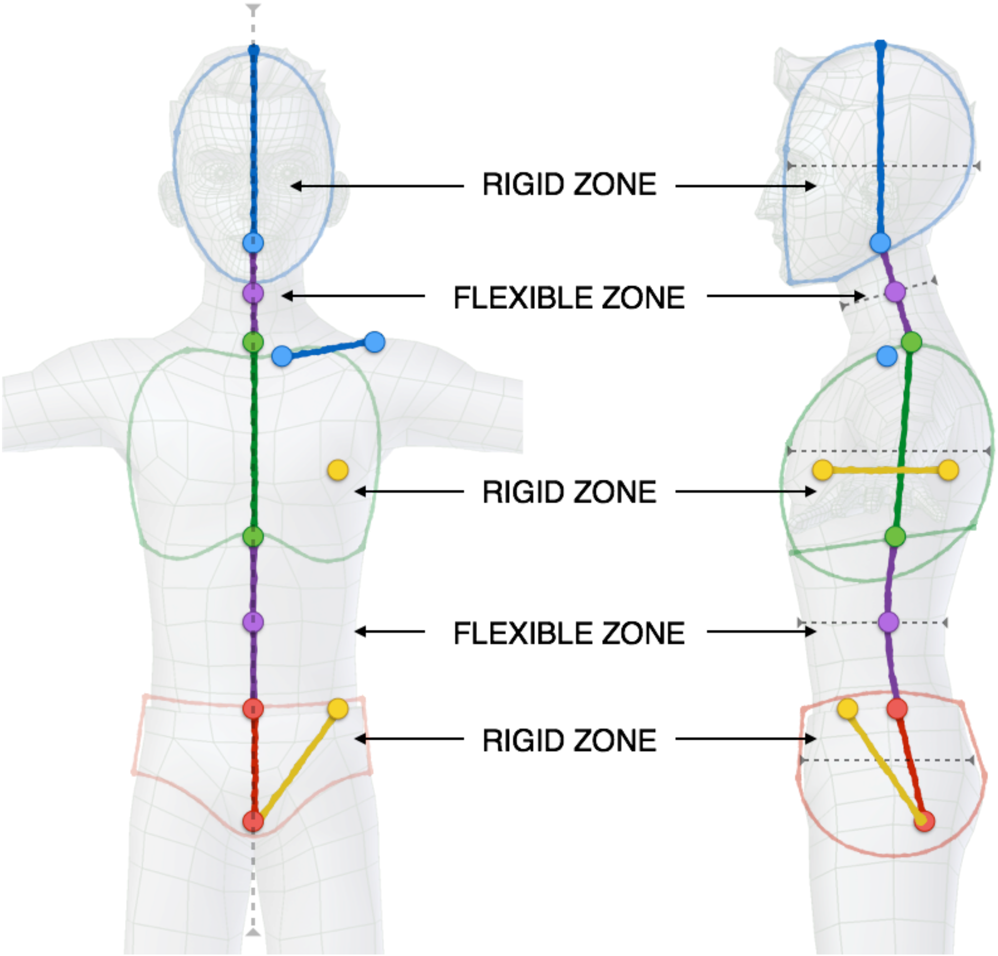
Head, chest and pelvis are rigid zones, so they require less bones. Having a good edge loop placement around zone boundaries on your model will help in having correct deformation after armature binding.



Torso Landmarks.

Starting from the side view, place the main spine bones trying to use one bone for the rigid areas and two for the flexible ones. In addition to the main spine, the torso is provided with additional pelvis bones (to oppose the leg bending), two breast controls and two shoulder bones.

Even if the pelvis bones will not appear in the final rig as controls, they will contribute to deformation.


RIGID ZONE
FLEXIBLE ZONE
RIGID ZONE
FLEXIBLE ZONE
RIGID ZONE

Torso Bones Positioning.

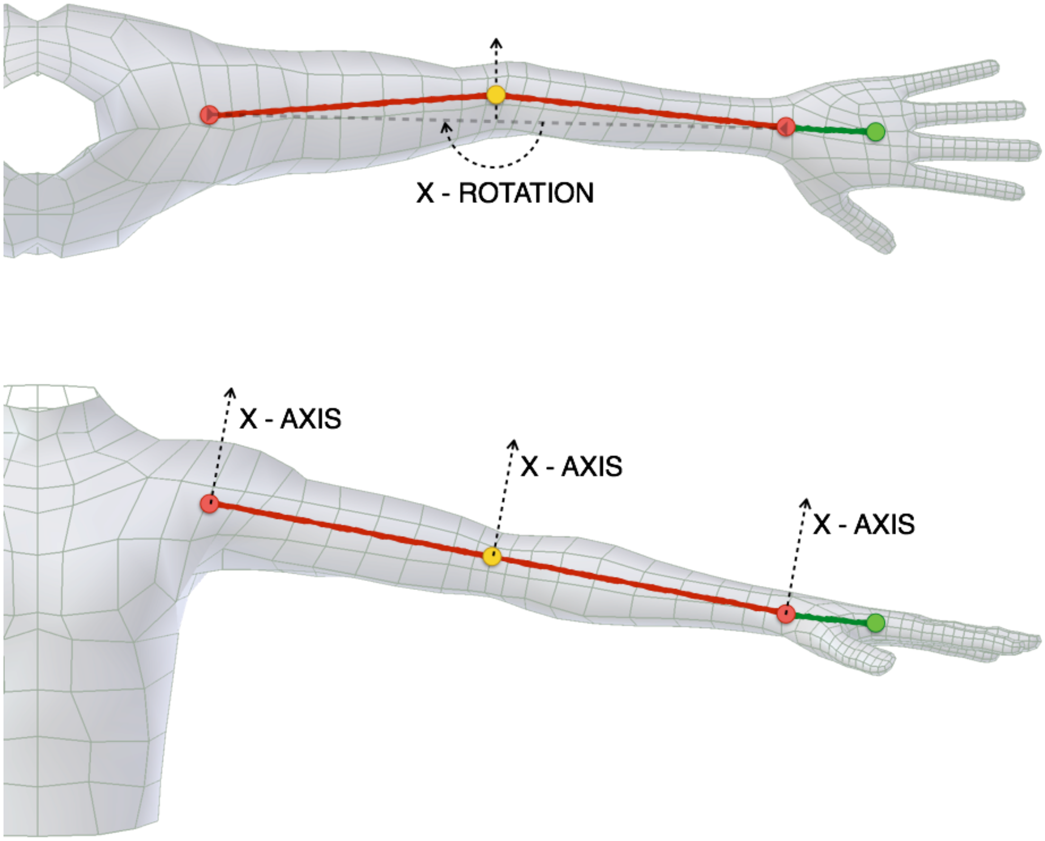
Tip

Bone Placement

Try to keep the spine as centered as possible inside the mesh bounding volume, just apply a slight offset toward the back. In a similar way, consider the shoulder bones as general deformers; placing it too forward – where the collar bone should be – could cause undesired deformations.

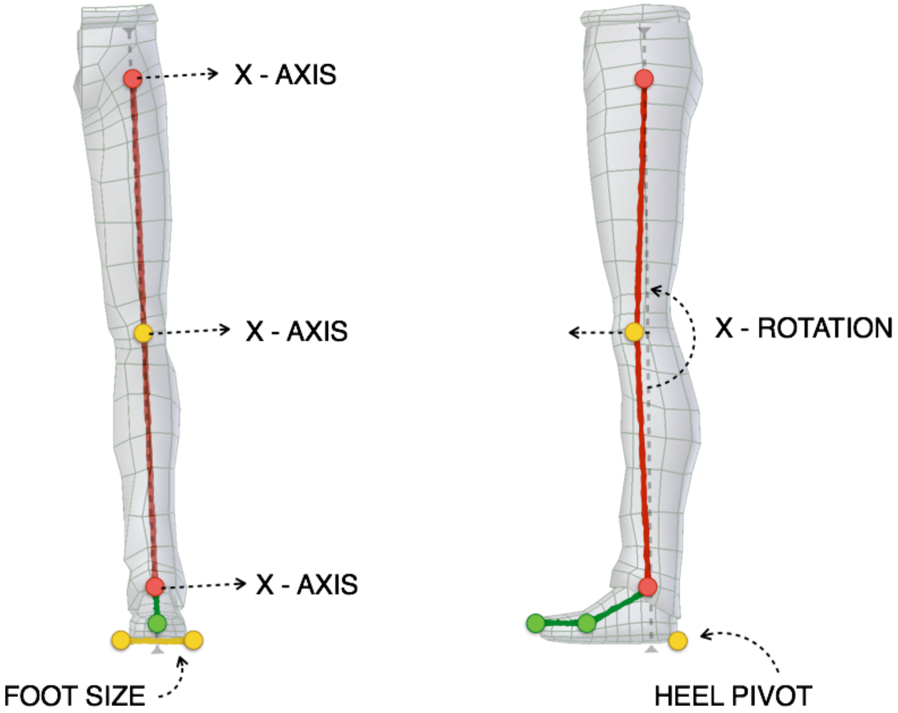
Limbs Bones

While placing the arm bones try to start having a straight line that goes from the shoulder to the hand in both front and top view. After this is done just add a slight bend to the elbow. This can be easily done by going in the top view, entering armature Edit Mode and sliding the bone junction between forearm and upper_arm slightly toward the world's Y axis.



Arm Bones Positioning.

For the leg you can follow a similar process. Start by aligning the leg bones creating a straight line from the hips to the ankle, then place the foot and the toe accordingly. Remember to add a slight bend to the knee. This can be easily done by going in the side view, entering armature Edit Mode and sliding the bone junction between thigh and shin slightly toward the world's Y axis.



Leg Bones Positioning.

Finally align the heel bone by going in the front view and placing the head and tail to fill the foot size from side to side. Then, in the side view, align the bone at the point where the heel just touches the ground floor.

Note

From version 0.5 and above there is no more need of manual bone rolls alignment. The generate function will take care of that for you by evaluating it from bend axis; just insert a slight bend in your limb and it's done! If you need more control on the orientation, follow the guidelines described in Advanced Usage.

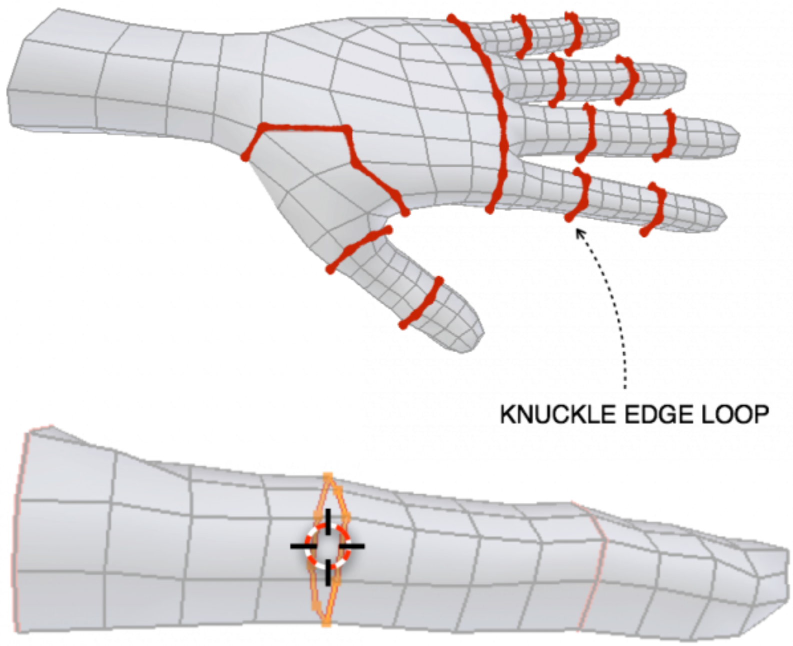
Fingers Bones

Start by placing, finger by finger, all the knuckles in place.

Tip

Fingers Placement

An easy and effective method to do this operation is to select on the mesh the corresponding edge loop in Edit Mode and use the *Cursor to Selection* snap. Then you can snap the bone to the corresponding loop using the *Selection to Cursor* snap.



Knuckles Edge Loops and Cursor Snapping.

Finalize the positioning by taking care of bone rolls (the X axis is set as bend axis).

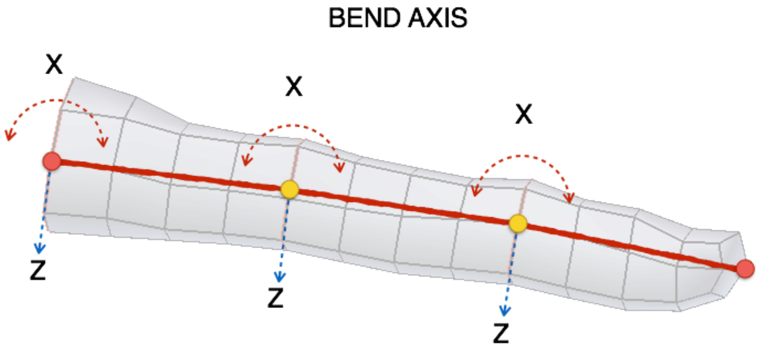
Tip

Bone Roll

Finger axis alignment can be easily be made consistent by selecting all the finger bones and recalculating the bone rolls *Recalculate Roll ▸ Global -Z Axis*.

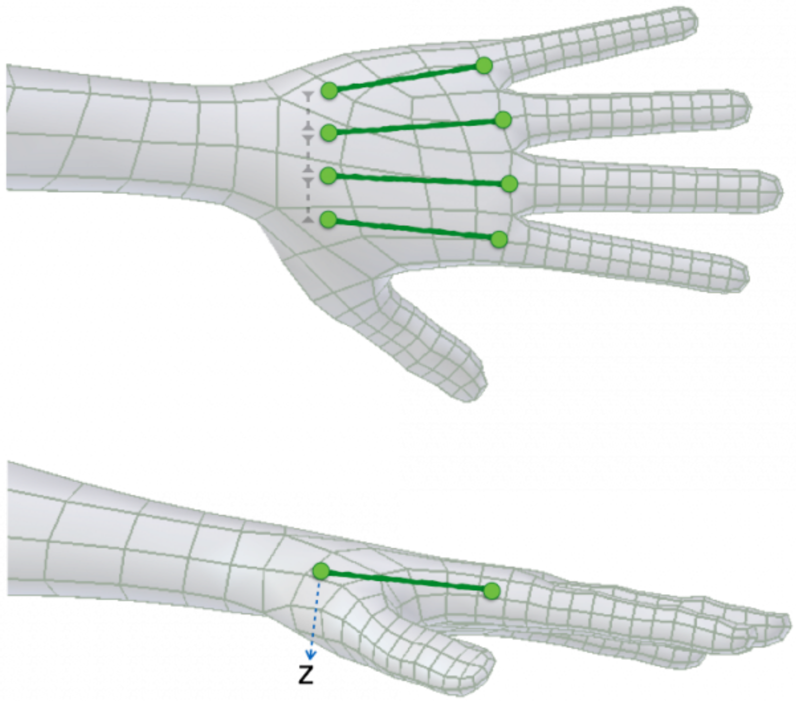
Thumb may require more tweaking depending on your character's mesh topology, usually *Recalculate Roll ▸ Global +Y Axis* is a good starting point.

Once your bone rolls are consistent, try generating the rig and scaling the finger master controls. This should cause the fingers to curl. If they are rotating on the wrong axis, change the Bend Rotation Axis parameter on the first finger's parameters under Rigify Type.



Fingers Bend Axis.

When the fingers are in place proceed placing the palm bones.



Palm Alignment.

Tip

Palm Placement

Try to keep palm bones’ heads at a little distance between each other. This distance is required for Rigify to define the palm controls hierarchy. Palm axis alignment can be easily done by selecting all the palm bones and recalculating the bone rolls *Recalculate Roll* ▸ *Global -Z Axis*.

See also

For more detailed information on bones and rolls refer to the Bone Structure and Bone Roll.

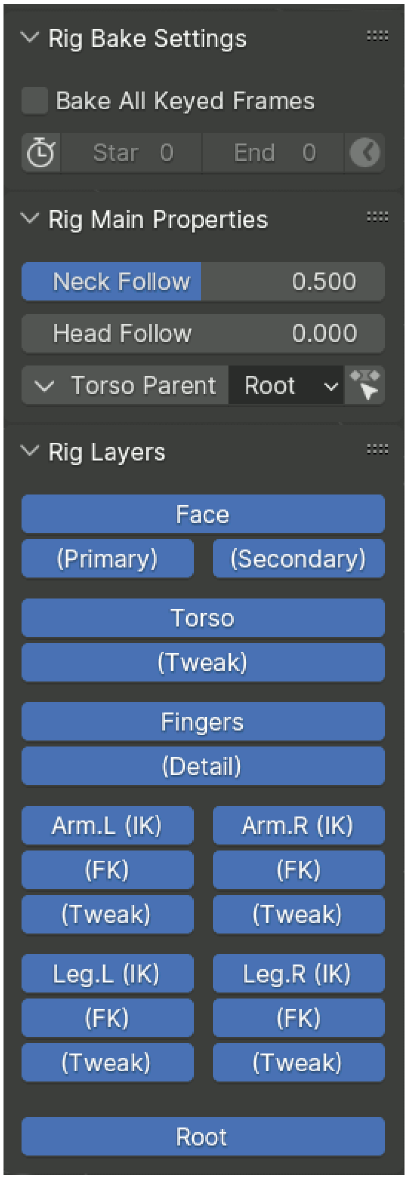
Generated Rig Features

After human rig generation a new armature named `rig` will be added to your scene. This is the character rig you have generated from the human meta-rig and will contain all the features.

Common Features

Rig UI Panels

The generated rig is accompanied by a script that implements a set of panels that appear in the Item tab of the 3D view sidebar when a bone belonging to the generated rig is active.


Rig Bake Settings
☐ Bake All Keyed Frames
Star 0
End 0
Rig Main Properties
Rig Layers
Face
(Primary)
(Secondary)
Torso
(Tweak)
Fingers
(Detail)
Arm.L (IK)
Arm.R (IK)
(FK)
(FK)
(Tweak)
(Tweak)
Leg.L (IK)
Leg.R (IK)
(FK)
(FK)
(Tweak)
(Tweak)
Root

Neck Follow 0.500

Head Follow 0.000

Torso Parent Root

Rig Bake Settings

This panel is displayed if the armature has an active Action, and is used by operators that apply an operation to multiple keyframes.

Bake All Keyed Frames

> When enabled, the operator computes and keyframes its result on every frame that has a key for any of the bones, as opposed to just relevant ones.

Limit Frame Range

When enabled, the operator is limited to a certain frame range.

Start, End

Specify the frame range to process.

Get Frame Range

Sets the baking frame range from the scene frame range.

Rig Main Properties

This panel shows properties and operators that are relevant to the selected bones.

Rig Layers

This panel contains buttons for toggling visibility of bone collections.

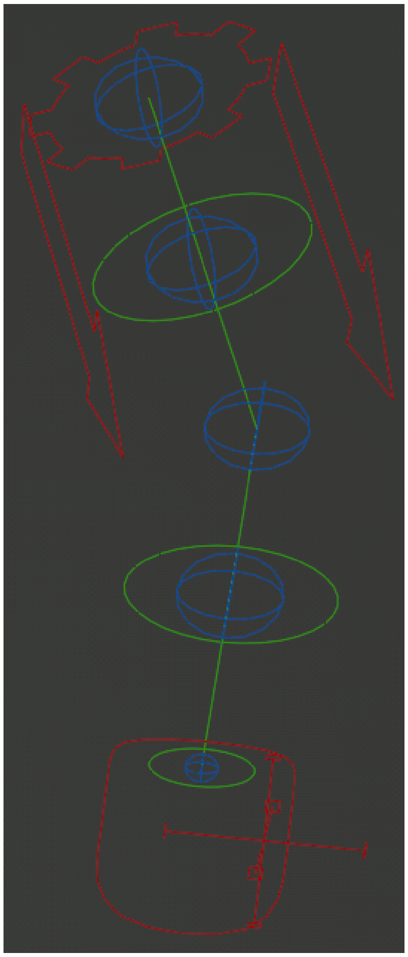
The layout and labels of the buttons are defined in the metarig Bone Collection UI panel.

Common Controls

Rigify rigs are built from standardized components called sub-rigs, which are linked together in a parent-child hierarchy. Although the precise behavior of each sub-rig is determined by its implementation, there are certain conventions that are followed by many of them.

Root Bone

Every Rigify rig has a bone called *root*, which serves as a parent for all bones of the rig. It is assigned to a bone collection called *Root*. Unless the metarig has a custom bone of that name, it is positioned at the origin of the rig object. Its widget looks like a circle with four arrow shaped protrusions.



Limb Master

Many limb-like sub-rigs have a gear-shaped bone at their base.

This bone can in some cases be used to transform the whole sub-rig as a rigid unit, and is also used as a container for its custom properties that are displayed in the *Rig Main Properties* panel. If you are looking in the Graph editor for the animated values of the properties, this is most likely the bone to look at.

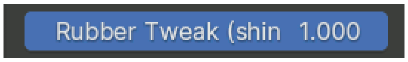
As an exception, if multiple controls of the sub-rig need their own copy of conceptually the same property, it may be placed on those controls directly instead.

Tweak Controls

These controls look like blue spheres in the default color scheme, and are the final control layer above the deformation bones themselves.

Tweaks are subordinate to the general IK or FK limb position but can be moved apart, twisted and scaled freely, even reaching virtually impossible limb shapes.

Rubber Tweak


Rubber Tweak (shin 1.000

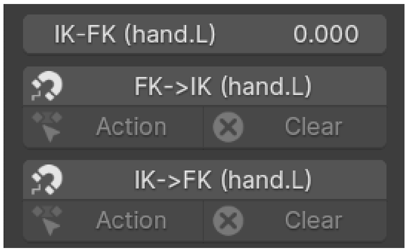
Some sub-rigs provide a slider in their *Rig Main Properties* when tweaks are selected, which controls the smoothness of the Bendy Bone joint at that position. When zero, the joint deforms with a sharp bend, while setting it to 1 makes the transition smooth for a more rubber hose cartoon like appearance.

Custom Pivots

Some bones that can be freely moved in space (like IK controls) can be optionally accompanied by a custom pivot control. These controls usually look like a plain axes empty with the axis lines capped with squares or crosses, like the one in the image above. The control can be freely moved to change the location of the pivot, and then rotated or scaled to transform the target bone around the pivot.

IK and FK Switching

A number of rig types provides both IK and FK controls (red for IK and green for FK in the image above), with an ability to switch and snap between them.

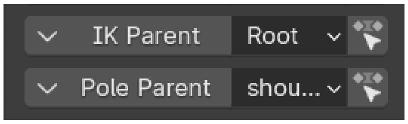

IK-FK (hand.L) 0.000

Switching is controlled by a slider in *Rig Main Properties*, usually blending between full IK at 0 and full FK at 1.

Snapping one type of controls to the shape of the other is done via buttons, which form a group of three in their complete set:

- The main button will snap on the current frame, and auto-key the result if enabled.
- The *Action* button will bake the change on multiple keyframes, according to *Rig Bake Settings*.
- The *Clear* button will delete keyframes on the corresponding controls within the bake interval.

Parent Switching



Some freely movable controls, e.g. usually the IK controls, can have a mechanism to switch their parent bone between a set of choices, including the root bone, or none at all.

This mechanism is exposed in the *Rig Main Properties* panel through a row with three controls:

- A button that presents a dropdown menu, which allows switching the parent on the current frame while preserving the bone position and orientation in the world space.
- A dropdown input field that directly exposes the switch property for keyframing and direct manipulation. Changing the value can cause the bone position to jump.
- A button to apply the position preserving parent switch over the bake range of keyframes.

Note

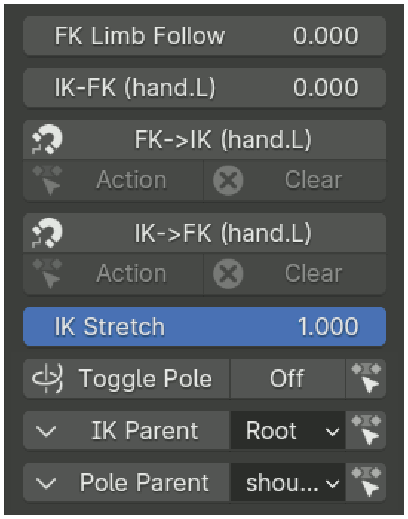
When manually placing a Child Of constraint on the control bone, the built-in parent should be switched to none.

Limbs

Limbs have a master bone and tweaks. Depending on the user defined meta-rig options, multiple deform bone segments with tweaks will be created.

The IK control may have an optional custom pivot, as well as additional predefined pivots.

Rigify’s limbs have the following controls in the Sidebar panel:



FK Limb Follow 0.000

IK-FK (hand.L) 0.000

IK Stretch 1.000

Toggle Pole Off

IK Parent Root

Pole Parent shou...

FK Limb Follow *Slider*

When set to 1 the FK limb will not rotate with the torso and will retain is rotation relative to the root bone instead.

IK-FK *Slider*

Controls whether the limb follows IK or FK controls, blending between full IK at 0 and full FK at 1.

IK<->FK Snapping *Buttons*

Snaps one type of controls to another.

IK Stretch *Slider*

Blends between the limb stretching freely at 1, or having its maximum length constrained at 0.

Toggle Pole *Switch*

When the toggle is Off, the IK limb will use the rotational pole vector (the arrow at the base of the limb). Rotating/translating/scaling the arrow will control the IK limb base.

When the toggle is On, the classic pole vector will be displayed and used to orient the IK limb. The arrow will continue to handle the scale and the location of the IK limb base.

Similar to *Parent Switching*, the row includes buttons to convert the current pose between types, or bake the whole action.

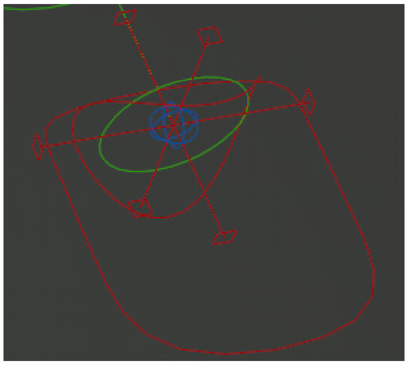
IK Parent *Switch*

Switches the effective parent of the main IK control.

Pole Parent *Switch*

Switches the effective parent of the classic IK Pole control.

Arms

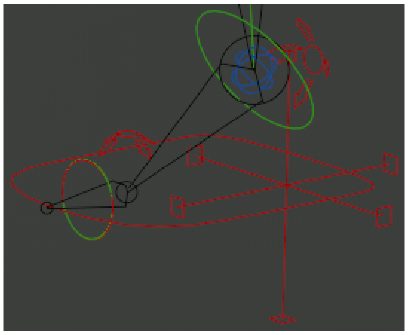


Arms have the simplest control structure: the IK controls consist of the main IK control, the optional custom pivot control, and the optional wrist control (the bent circle), which pivots around the tail rather than the head of the hand bone.

There are no additional controls in the *Rig Main Properties* panel.

Legs

Legs have a more complicated setup, which has:



IK & FK Toe *Optional*

Two separate IK and FK controls for the toe (this is on by default in the bundled metarigs, and is recommended for stable IK<->FK snapping).

IK Heel

A heel control which can be rotated to command forward or backward roll, sideways rock, or yaw of the heel.

Toe Pivot *Optional*

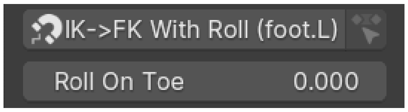
An extra pivot control rotating around the base of the toe.

Custom Pivot *Optional*

A custom pivot control.

The properties panel has two additional features:

IK->FK Snap With Roll *Buttons*



Roll On Toe 0.000

Standard IK to FK snapping resets the transformations of all IK controls other than the main one. This is not convenient to use in an animation that involves the use of the heel control, because roll and rock would be folded into the transformation of the main control.

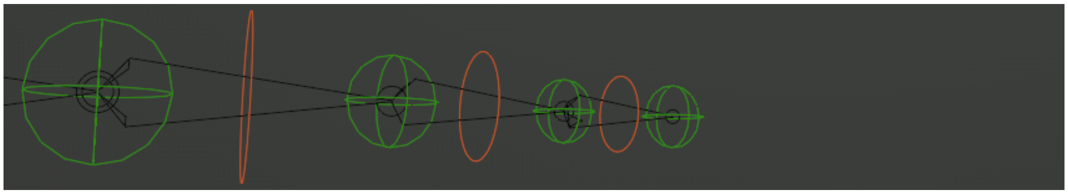
This alternative snapping operator tries to deduce the rotation of the heel control so as to keep the main IK control parallel to the ground plane inferred from the *current* orientation of the IK control. The operator has options to specify which rotational axes to use for the heel control rotation.

Roll On Toe *Slider Optional*

If enabled in the sub-rig settings, this slider can be used to control whether the heel rotation (excluding backward roll) is applied at the base or the tip of the toe.

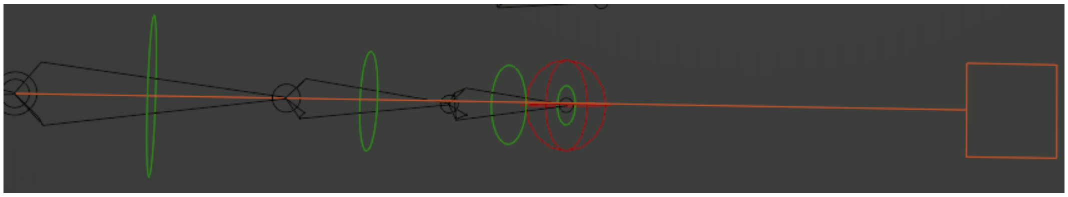
Fingers & Tentacles

Simple Tentacle



The simplest type of rig for a finger or appendage in general is the simple tentacle sub-rig. It has only basic FK controls and tweaks, with the only automation being the ability to copy certain axes of the local rotation of a FK control to the next one.

Advanced Finger



For fingers specifically, Rigify has a dedicated finger sub-rig type, which provides:

Master

A master control (orange), which can be used to rotate the finger as a whole, as well as to bend it via Y scaling.

FK Chain

FK control chain (green) that can also operate as semi-tweaks through allowing translation.

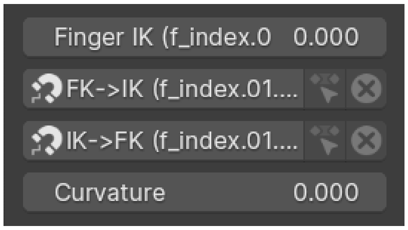
IK Control *Optional*

IK control for the tip (red).

Note

IK in this sub-rig is rudimentary and operates as an adjustment for FK. The intended way of use is to pose the finger in FK, and then enable IK after using IK->FK snap if it is necessary to pin the tip of the finger in place.

The properties panel has the following features:



Finger IK (f_index.0 0.000

FK->IK (f_index.01....

IK->FK (f_index.01....

Curvature 0.000

Finger IK *Slider Optional*

Slider controlling the influence of the IK.

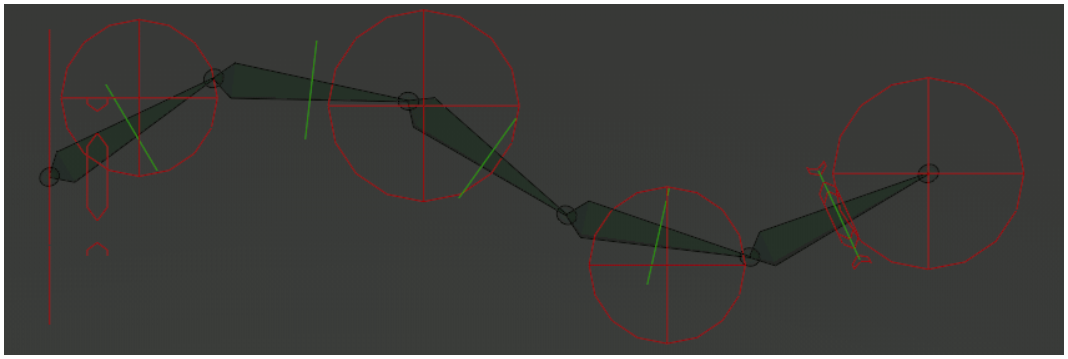
FK<->IK Snapping *Buttons Optional*

Snaps the IK control to the end of the finger, or adjusts the FK controls to the result of the IK correction.

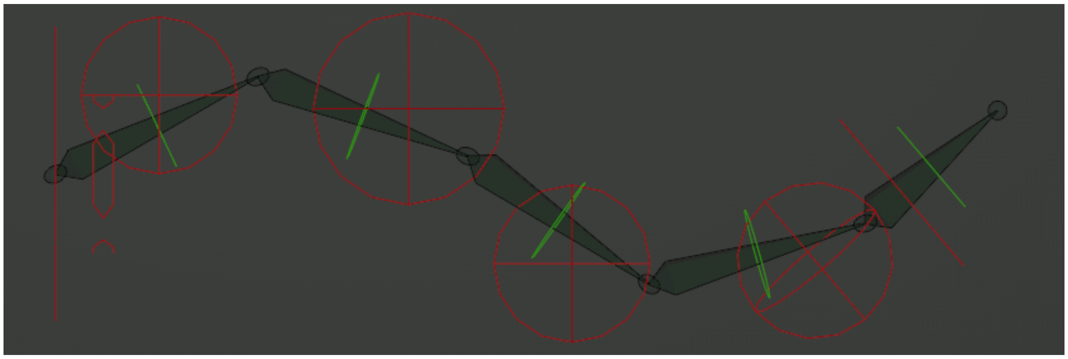
Curvature *Slider*

Has the same effect as *Rubber Tweak* on limbs, controlling the rubber hose cartoon effect.

Spline Tentacle



Spline Tentacle (Stretch To Fit, Manual Squash & Stretch)



Spline Tentacle (Direct Tip Control)

The spline tentacle is an advanced rig for a flexible appendage (tentacle) based on the Spline IK constraint. The IK control bones manage control points of a Bezier spline curve, which in turn is followed by the IK chain.

The tentacle can be generated in three major modes:

Stretch To Fit
: In this simplest mode all bones of the sub-rig deform chain follow the curve and squash & stretch to match its length.

Manual Squash & Stretch

This mode is almost the same, but the chain does not automatically scale to match the curve length. Instead, it tries to cover as much as possible of the curve given its manually scaled length. If the curve is too short, the chain will overhang it and straighten out, but this can result in jitter.

Direct Tip Control

This mode is more similar to the behavior of IK limbs: the final bone of the chain is directly controlled by the tip IK control, while the other bones of the chain stretch and follow the curve to bridge the gap.

The tentacle sub-rig has the following control bones:

Master

The tentacle has the same gear master control as other limbs (seen as a line in the images).

IK Start

The IK control at the base of the tentacle, which can be used to control the base twist and sideways scale, and is one of the potential switchable parents for other IK controls.

In the *Manual Squash & Stretch* mode it controls uniform scale of the tentacle in all directions.

IK Start (Extra) *Optional*

Extra start controls, optional and hidden by default. Switchable parents default to the *IK Start* control. The scale of the control may optionally affect the thickness of the chain via the radius of the curve point.

IK Middle

Controls for the middle of the curve. The switchable parents default to *Master*, but may be set to *IK Start* or *IK End* controls. The scale of the control may optionally affect the thickness of the chain via the radius of the curve point.

IK End (Extra) *Optional*

Extra end controls, optional and hidden by default. Switchable parents default to the *IK End* control. The scale of the control may optionally affect the thickness of the chain via the radius of the curve point.

The *Direct Tip Control* mode adds one more extra end control next to the middle ones that cannot be hidden.

IK End

Controls the last control point of the curve, and is one of the potential parents for the other chain controls.

In the *Direct Tip Control* mode also directly controls the last bone of the chain.

IK End Twist *Optional*

This control is visually attached to the last bone of the chain, and must use Euler rotation.

- *Stretch To Fit*: it controls the twist of the tip of the tentacle, interpolated to nothing at the base.
- *Manual Squash & Stretch*: it also controls the scaling of the tip of the tentacle.
- *Direct Tip Control*: the control does not exist.

FK Chain *Optional*

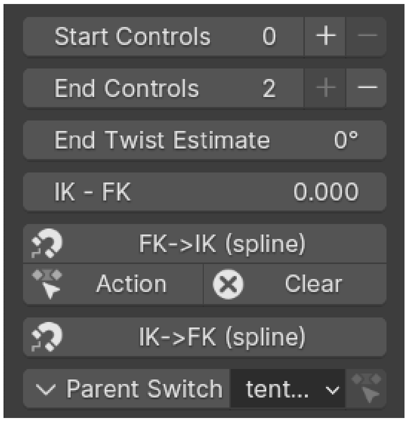
If enabled, the rig has an alternative fully FK control chain.

The properties panel has the following features:

Start/End Controls *Optional*

If extra controls exist, this property controls how many of them are visible and active.

When a control is disabled, it is snapped to a position extremely close to the corresponding end control point, thus effectively neutralizing its effect. Thus, changing the setting during an animation can cause jumps.


Start Controls 0
End Controls 2
End Twist Estimate 0°
IK - FK 0.000
Parent Switch tent...

The plus and minus buttons can help with maintaining a continuous transition in an animation by keyframing the change in the property value with Constant interpolation, and also snapping and keyframing the control itself to its ‘hidden’ position.

End Twist Estimate *Optional*

In the *Direct Tip Control* mode the twist at the end of the tentacle is deduced from the free form orientation of the tip control, rather than using a separate twist control with constrained Euler rotation. However, for technical reasons, that can only give values within the 180 degrees range of neutral.

A long tentacle can accept more twist than 180 degrees, so a workaround is necessary. This property allows specifying an approximate estimate of the twist value (effectively shifting the neutral position), and the rig then applies the automatic correction within 180 degrees of this value.

IK-FK, IK<->FK Snapping *Optional*

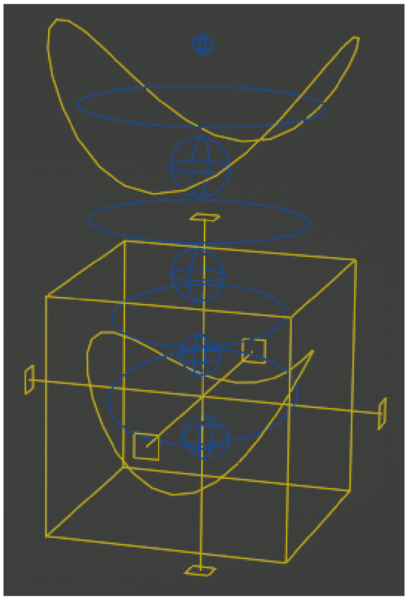
If the FK controls are enabled, these provide standard IK-FK switching and snapping.

However, unlike other limbs, for this rig automatic IK to FK snapping can only be approximate and requires manual tuning. For this reason, buttons for baking the snapping over a range of keyframes are not provided.

Parent Switch

Switches the parent of the selected IK control.

Spine, Head & Tail

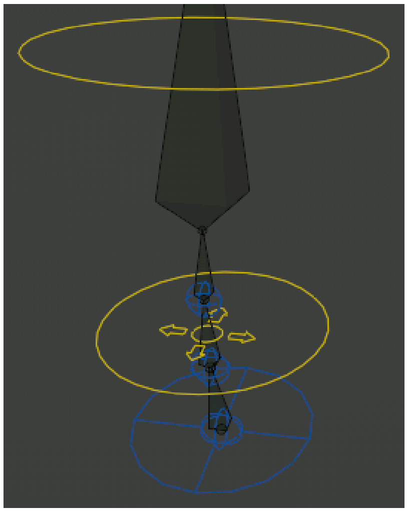


Spine

The spine sub-rig provides a cube shaped torso control with switchable parent, and bent circle shaped hip and chest controls subordinate to it. For low level deformation tweak controls are provided.

The torso control can optionally be accompanied with a custom pivot control. The rig can also optionally provide a full set of FK controls that are subordinate to the normal simplified ones, but above tweaks.

The rig properties panel for the spine controls usually includes options for the head and/or tail as well.



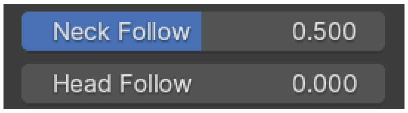
Head

The head sub-rig attaches to the end of the spine, and provides rotational controls for the head and neck, as well as tweaks for fine control of the neck.

If the neck is three or more bones long, an additonal tweak-like translational neck bend control is provided (the widget looks like a circle with arrows).

The properties panel contains the following options:

Neck Follow *Slider*

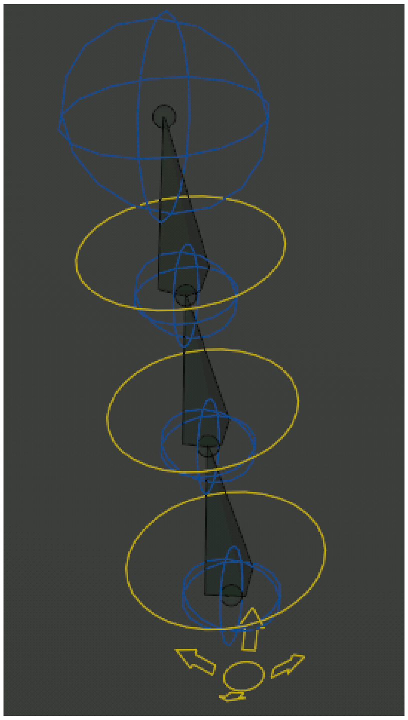


Neck Follow 0.500
Head Follow 0.000

> This slider controls the rotations isolation for the neck bones. The neck will follow the orientation of the Torso when set to 0, and the Chest when set to 1.

Head Follow *Slider*

> This slider controls the rotations isolation for the head. The head will follow the orientation of the Torso when set to 0, and the Chest when set to 1.



Tail

The tail sub-rig attaches to the start of the spine, and provides FK controls for the tail, as well as a master control that replicates its local rotation around certain axes to all individual bones.

The properties panel contains the following options:

Tail Follow *Slider*
: This slider controls the rotations isolation for the tail. The tail will follow the orientation of the Torso when set to 0, and the Hips when set to 1.

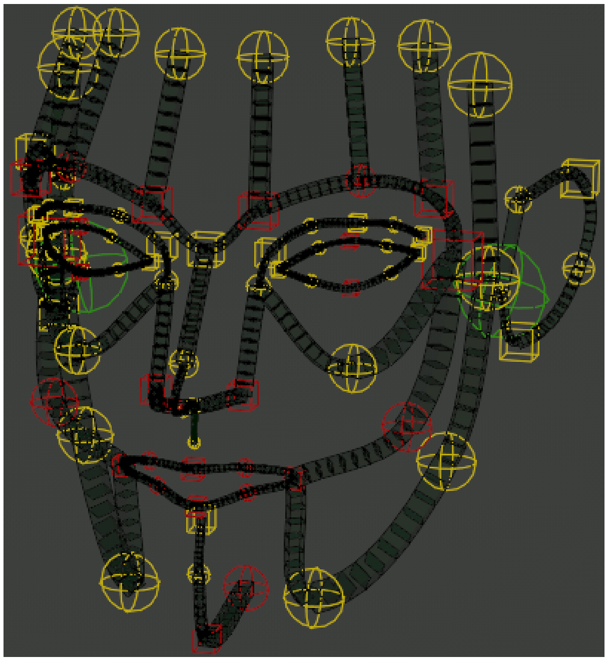
Face

Note

This describes the new-style modular face produced by the Upgrade Face operator button.

Basic Concepts

Skin Bone Chains



The foundation of the Rigify face is a network of Bendy Bone chains with controls placed at every bone end. These controls affect all bones that meet at that specific point.

When the controls are merely translated, the B-Bone chains retain the normal automatic bezier handle behavior. Local rotation and/or scaling of the controls are applied on top of that.

In case of certain chains, the transformation of the end and/or middle controls is interpolated to other controls located between them. In such cases the controls often have different colors and/or shapes.

Additionally, certain controls have arbitrary constraints that partially copy transformation from nearby control points.

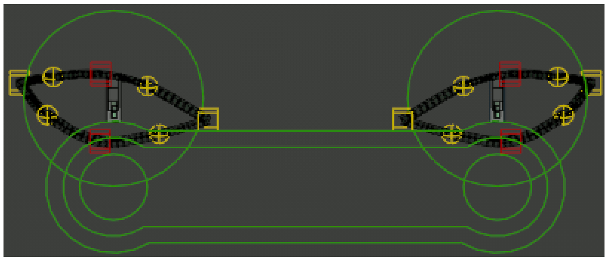
Specialized Controllers

Certain areas of the face, like eyes or mouth, have additional specialized controllers that apply custom behavior on top of the chains and their controllers within the relevant area.

Eyes

The eyes have the following controls in addition to the eyelid chains:

Master



This large circular control can be used to transform the whole eye as one unit.

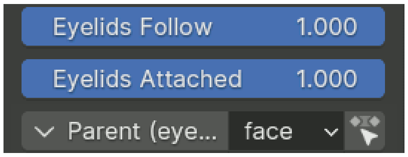
Common Target

This large control enveloping all individual eye targets has a switchable parent and can be used to specify the point that the eyes should look at.

Eye Target

These small circle controls within the common target control specify the point targeted by each individual eye. Their local scale can also be used to affect the iris or pupil of the eye, depending on how it was weight painted.

The rig properties panel contains the following options:



Eyelids Follow 1.000

Eyelids Attached 1.000

Parent (eye... face

Eyelids Follow *Slider*

Controls how much the rotation of the eyeball affects the eyelids. Depending on the sub-rig generation options, this slider can be split to separately control the horizontal and vertical directions.

Eyelids Attached *Slider Optional*

If enabled in the sub-rig generation options, this slider can be used to disable the mechanism that forces the eyelids to conform to the sphere of the eye.

Parent *Parent Switch*

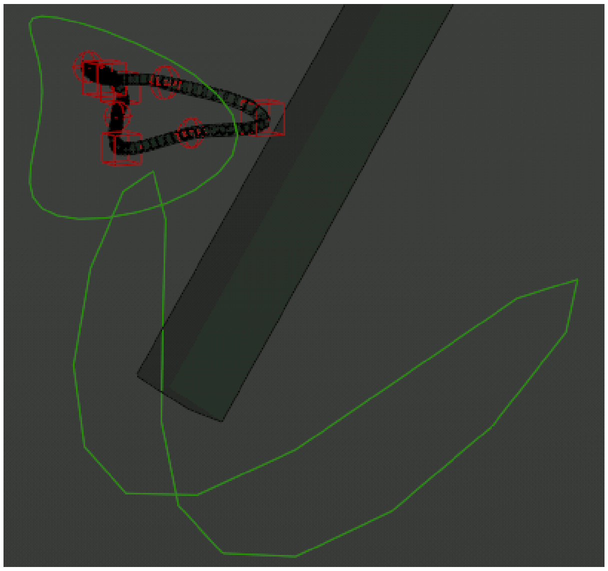
Selects the parent for the common target control.

Mouth

The mouth has the following controls:

Jaw Master

Controls rotation of the jaw, directly affecting the main jaw deform bone, as well as chains fully belonging to the jaw. Chains forming the lip loop(s) are adjusted to open the mouth as the jaw rotates or moves.

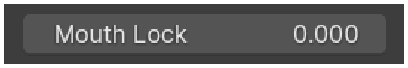


Mouth Master

This control uniformly transforms the lips without moving the jaw.

The rig properties panel contains the following options:

Mouth Lock *Slider*



Mouth Lock 0.000

This slider can be changed from 0 to 1 in order to suppress opening of the mouth when the jaw rotates or moves.

Creating Meta-rigs

1. Add a single bone from the *Add ‣ Armature* menu.
2. Go in armature Edit Mode and build the meta rig by samples or Rigify-types.
3. Define the Rigify bone collection UI, color sets, and selection sets.
4. In the armature properties click on the *Generate* button to generate the rig.

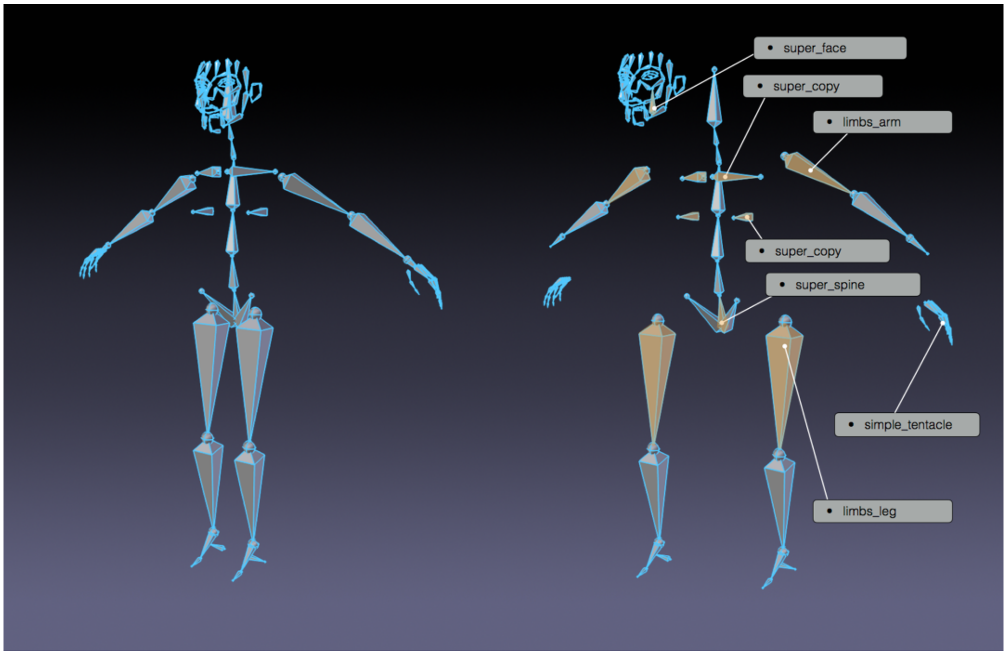
How Rigify Works

Rigify Meta-Rigs are split in multiple Sub-Rigs

> A meta-rig is an assembly of bone chains. A bone chain is identified by the *Connected* attribute. Bone chains can be further connected together by parenting them without using the *Connected* attribute (i.e. using the *Keep Offset* option while parenting).

A custom attribute is set on the first bone of the sub-rig chain

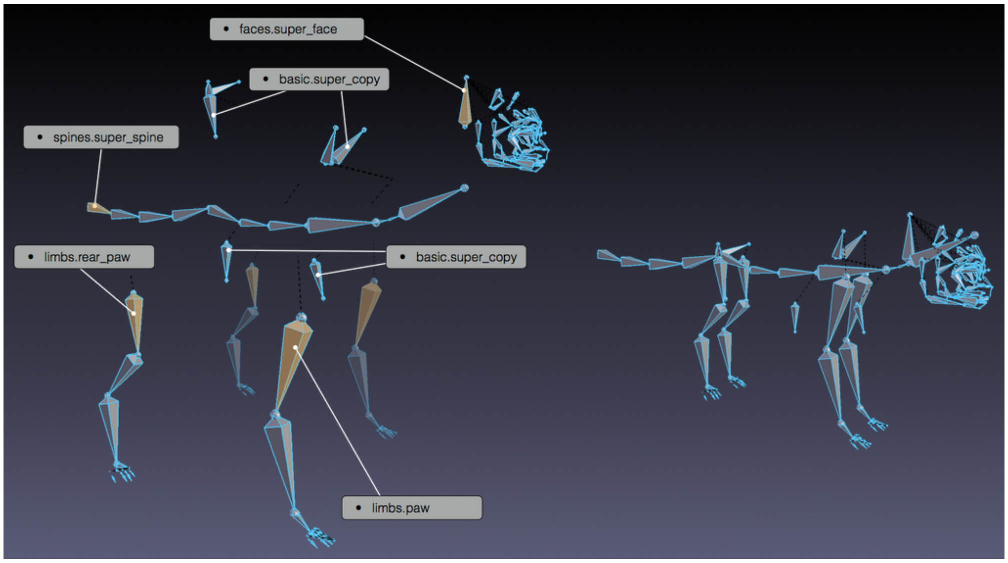
> Each first bone of a bone chain has a custom attribute on it which is a Rigify custom property that identifies the sub-rig type. At rig generation time Rigify will determine which controls and deform bones will be created processing the meta-rig from the first bone to the last of each chain.


super_face
super_copy
limbs_arm
super_copy
super_spine
simple_tentacle
limbs_leg

Human meta-rig split by samples.

New meta-rigs are created assembling sub-rigs samples

Since a meta-rig is just a collection of sub-rigs, new meta-rigs can be built assembling sub-rigs in different ways. This way an infinite number of meta-rigs can be built from the same rigging blocks.


faces.super_face
basic.super_copy
spines.super_spine
limbs.rear_paw
basic.super_copy
limbs.paw

Cat meta-rig built by samples.

All the mechanics, deformation bones and widget are created on a single click

> The meta-rig contains more information than the visualized bones. In fact at generation time Rigify will identify each sub-rig type and depending on the selected options will create all the sophisticated controls, switches, and deforming bones with a single click.

Creating a new Meta-rig

Add a new Armature Object

Reference

Mode::	Object Mode
Menu::	*Add* ‣ *Armature* ‣ *Single Bone*
Shortcut::	`Shift-A`

Building your own meta-rig from scratch requires an armature object to work with. Just add a single bone from the *Add* menu.

Tip

At this stage naming the newly added armature `metarig` is a good idea. You can do it at any time (or not at all) but it's suggested to do it before going on so it will always be clear on which armature you have to work when editing the meta-rig structure.

Editing the Armature

Now that there is an armature object to work – with the armature selected – enter armature Edit Mode. Building a meta-rig from scratch in Edit Mode can be done in two ways:

1. Adding rig samples.
2. Creating bone chains.

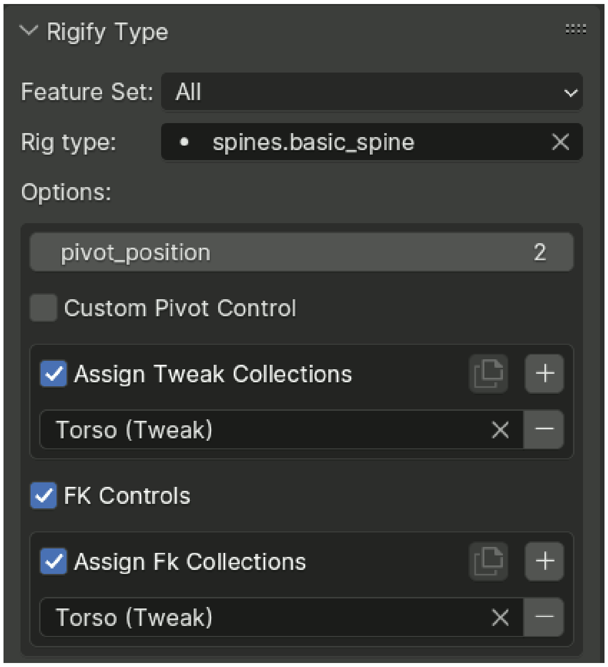
Adding Samples (Basic)

Adding pre-defined samples in Edit Mode is a good way to start building a meta-rig. This way you can become familiar with the available building blocks and how they are meant to be used. To add a rig sample:

1. Go in the armature tab.
2. Scroll down to Rigify panel.
3. Select a sample from the list.
4. Click on the *Add sample* button.
5. Edit the bone positions to match your character.

For the list of available samples, see the Rig Types page.

Using Rig Types (Advanced)



Rigify Type

Feature Set: All

Rig type: spines.basic_spine

Options:

pivot_position 2

☐ Custom Pivot Control

☑ Assign Tweak Collections

Torso (Tweak)

☑ FK Controls

☑ Assign Fk Collections

Torso (Tweak)

For full control, you can use the Rigify Type panel of bone properties in Pose Mode to assign any Rigify sub-rig type to any bone, as well as change its options.

For the list of available sub-rig types and their options, see the Rig Types page.

At the top of the panel you can find a field specifying the rig type for the active bone. The drop-down list can be optionally filtered by the Feature Set it belongs to.

Below that you can change options relevant to the selected rig type, if it has any.

Bone Collection References

Some rig types that generate many control bones have options that reference Bone Collections. These reference lists have a standard UI with the following features:

- A checkbox controlling whether the reference should be used.
- A button to copy the reference list contents from the active to all selected bones.
- A plus button to add a new reference to the list.
- A list of references, each entry with a field to specify the target collection, and a button to remove the entry from the list.

Note

Each sub rig has a required number of bones as input. If you are unsure on how to use rig-types properties, add a rig sample to your armature to see how it is supposed to be used.

Preserved Bone Properties

Certain properties of the metarig bones are often copied to the generated rig control, deform and mechanism bones.

The exact set depends on the sub-rig and the specific generated bone, and the sub-rig may override some properties even when it preserves others from the same subset, but there are certain common patterns:

Parenting Settings

> This subset consists of the parent ORG bone, Use Connect, Use Inherit Rotation, Use Local Location, and Inherit Scale.
>
> It is usually copied to deform bones, FK controls, and in other cases where the sub-rig doesn’t have a reason to completely override them.

Bendy Bone Settings (Edit Mode)

> Consist of the segment count, Vertex Mapping Mode, Ease In/Out, Roll In/Out, Curve In/Out and Scale In/Out.

The segment count is often overridden via a sub-rig option, but other settings are usually copied to deform bones as is.

Transformation Settings

Consist of the rotation mode, pose mode rotation values, and channel locks.

These settings are usually copied to FK controls.

Custom Properties

Usually copied to one of the controls generated based on the metarig bone (mainly FK). Intra-armature drivers that access the property are retargeted to the copied instance.

Custom Widget

Usually copied to one of the controls generated based on the metarig bone (mainly FK), and suppresses automatic generation of a widget for the bone if specified.

Custom Root Bone

If the meta-rig contains a bone called `root`, it is used as the root control bone instead of creating a new one. This allows changing the rest position of the root bone, assigning a custom widget, or adding custom properties to the bone.

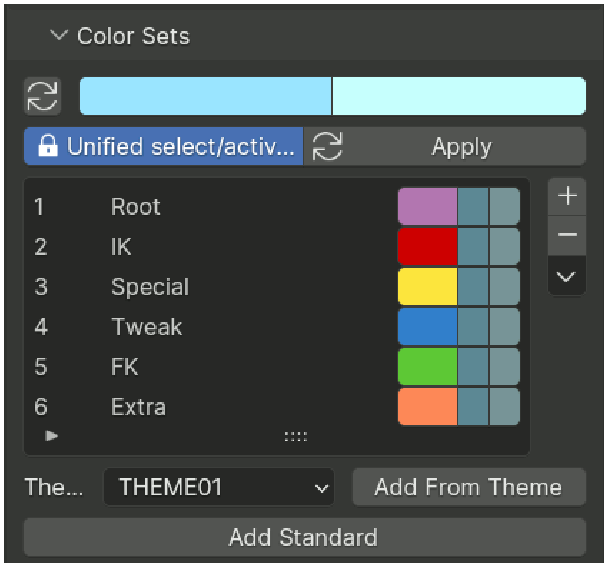
The custom root bone must have no parent, and use the basic.raw_copy sub-rig type or none.

Color Sets

The Color Sets panel is used to define the bone color scheme for the final rig. The colors from the list can be associated with bone collections from the relevant panel.

The top two rows of the Color Sets panel are used to define the general behavior of the bone colors. Usually color themes use a gradient of colors to define the different bone states: default, selected and active. When multiple color themes are used in the same rig, identifying which bone is selected or active can be tricky since each color will have its corresponding state.

To override this behavior Rigify unifies the active and selected states using the same color. This is defined by two values:



Unified Selected/Active Colors

> When this option is active adding a bone group in the list will always keep the colors consistent. When a color scheme is added from a theme, the color scheme is loaded as is. Click on the *Apply* button to force the system to unify selected and active colors.

Selected/Active Colors

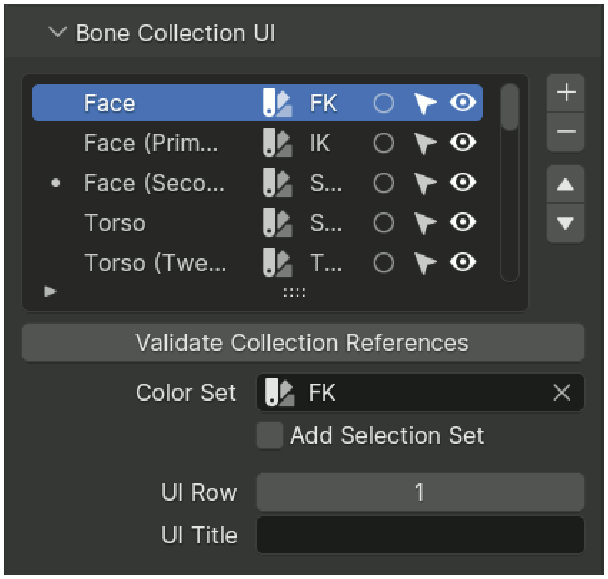
> This two color fields define respectively the *Selected* and *Active* colors. By default Rigify reads these colors from the theme defined by the user in the Blender preferences. This way the *Selected/Active* colors can always have a predictable and consistent behavior in the UI. The colors can be customized by clicking on the relevant color field. To reset them to the Blender current theme value just click on the button with the update icon.

Color Sets can be added and deleted by clicking on the + or – buttons. All color sets can be deleted at once by clicking on the Specials menu.

To add the colors from the predefined Rigify default color scheme (as shown in the image) to the list click the *Add Standard* button.

To add a specific theme with its own color scheme, select it from the list and click on the *Add From Theme* button.

Bone Collections UI



Color Set FK

☐ Add Selection Set

UI Row 1

UI Title

Bone Collections are used to group related bones together so that they can be hidden or revealed together.

Rigify can take advantage of collections to generate extra features and the user interface for the final rig. A panel named Rig Layers is generated with buttons for hiding the collections, arranged in an intuitive layout.

The Bone Collections UI panel allows configuring the layout of that generated panel, as well as specifying some other settings for bone collections, such as the color set to use.

The top of the panel is occupied by a list that duplicates the main bone collection list, but displays additional properties, such as the color set, whether the collection has a button, or whether it generates a selection set.

Validate Collection References

> Some sub-rig types have references to bone collections in their properties. Rigify uses a referencing scheme that is robust to collection renames, but deleting collections or joining armatures can still lead to broken references.
>
> This button runs a scan that validates and normalizes all collection references, reporting any errors, and reducing the chance of breakage being caused by subsequent user actions.
>
> This scan is also performed automatically every time the rig is generated.
>
> **Warning**

> To avoid breakage this operation should be used both immediately before and after joining two metarig armatures. More specifically, it must be always done between the actions of renaming any collections and joining.

Color Set
: Specifies the color set to use for bones in this collection. If a bone belongs to multiple collections, in general the collection located earlier in the list has priority.

Add Selection Set
: Specifies whether a selection set should be generated for this collection.

UI Row
: If nonzero, specifies which row of the Rig Layers panel should contain the button controlling the visibility of this collection. When zero, no button is generated, and the collection is hidden.

UI Title
: This field can be used to override the title used on the UI button to be distinct from the true collection name. Unlike collection names, titles are not required to be unique, so this can be used to reduce clutter by relying on contextual cues within the panel.

UI Layout sub-panel

The UI Layout sub-panel provides a WYSIWYG editor for the layout of the generated UI panel (as defined by the UI Row and UI Title settings above).

Each row contains three buttons at the end:

Arrow
: Moves the active collection button to this row.

Plus
: Inserts a new row before the current one.

Minus

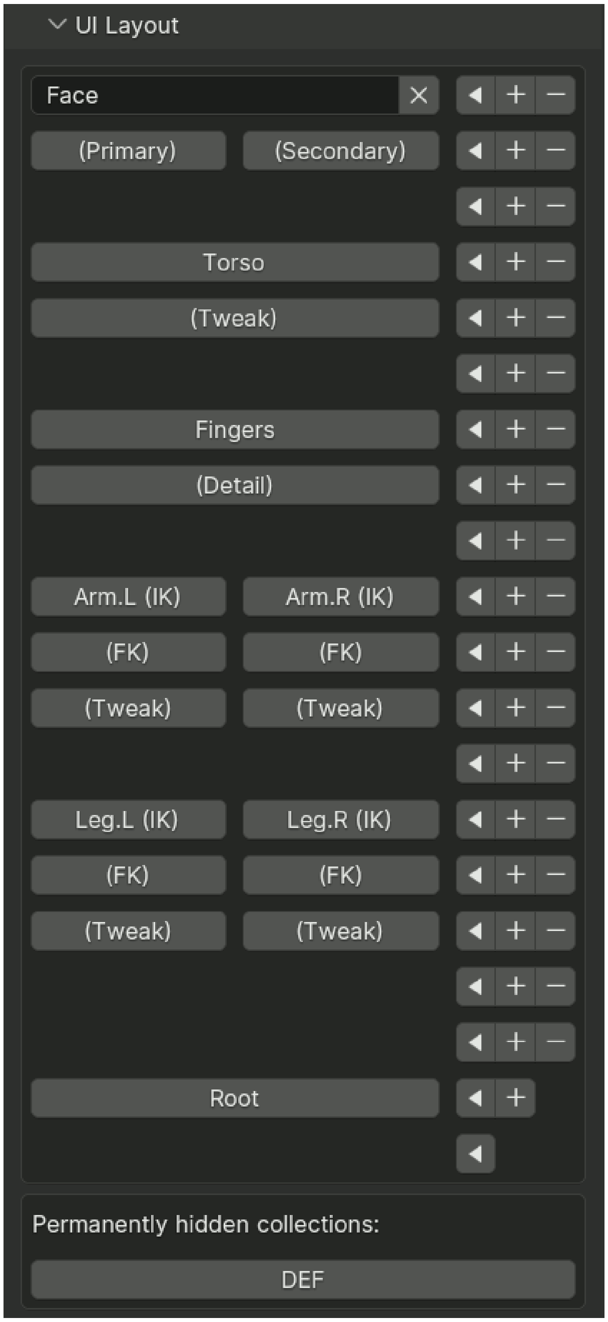
Removes the current row and shifts all buttons up.

To the left of the editing control buttons, rows display buttons corresponding to the collections, same as the final UI, except that rather than hiding or unhiding, clicking these buttons selects the collection.

For the active collection the selection button is replaced with an input field for editing the UI Title, and an **X** button to unassign the collection from the UI.

For any collections not assigned to the UI, their select buttons are displayed in a separate section at the bottom of the sub-panel.

The `Root` collection will be added and/or assigned a UI button automatically if necessary when the rig is generated. If desired, it is possible to manually assign UI buttons to the internal `ORG`, `DEF` and `MCH` collections.

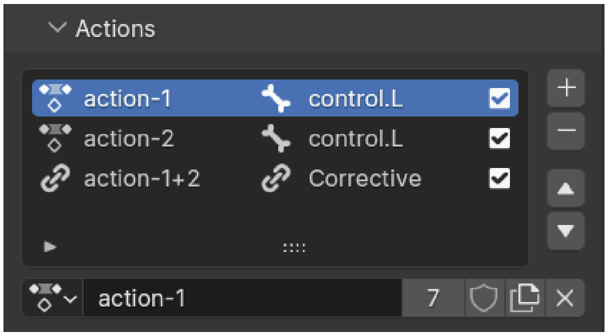

UI Layout
Face
(Primary)
(Secondary)
Torso
(Tweak)
Fingers
(Detail)
Arm.L (IK)
Arm.R (IK)
(FK)
(FK)
(Tweak)
(Tweak)
Leg.L (IK)
Leg.R (IK)
(FK)
(FK)
(Tweak)
(Tweak)
Root
Permanently hidden collections:
DEF

Tip

Blank rows appear much thinner in the final interface, since they don’t have to contain editing buttons, and can be used as logical separators.

Actions

The Action constraint allows applying poses defined by an action to bones based on the transformation of another bone. This requires adding the constraint to every bone affected by the action, which is very tedious. For this reason, Rigify includes a system to do this automatically through the Actions panel.



action-1	control.L	☑
action-2	control.L	☑
action-1+2	Corrective	☑

The panel defines a list of actions to be applied to the generated rig bones. Each action must be listed only once.

The list entries show the name of the action, the trigger (a bone or a corrective action driven by two others), and a checkbox that can be used to temporarily disable applying this action to the rig. The icon at the start of the entry is changed from an action icon to a link icon to highlight corrective actions that depend on the active normal one, or normal actions used by the active corrective action.

Note

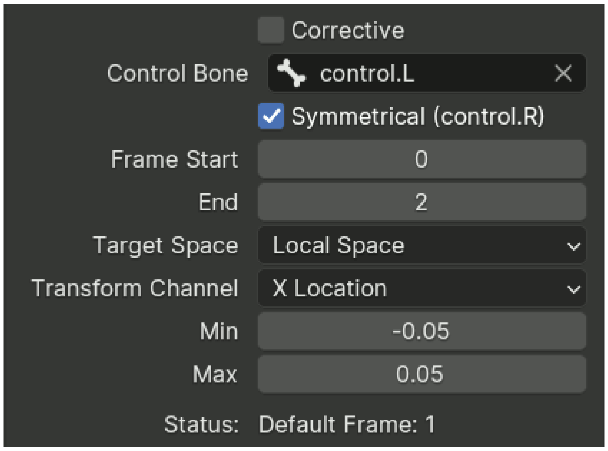
The Action constraints are added to the bones in such an order as to exactly reproduce the intended deformation, assuming the actions were created (posed and keyframed) in the order listed.

Normal Actions

Normal actions are applied based on the transformation of a specific control bone from the generated rig. They have the following properties:

Control Bone

Specifies the bone that drives the action.



☐ Corrective

Control Bone: control.L

☑ Symmetrical (control.R)

Frame Start: 0

End: 2

Target Space: Local Space

Transform Channel: X Location

Min: -0.05

Max: 0.05

Status: Default Frame: 1

Symmetrical

If the control bone has a suffix that specifies that it belongs to the left or right side, this option can be enabled to automatically apply symmetry.

When enabled, left-side bones keyframed in the action will be controlled by the left-side control, and right-side bones by the right side control. Bones that don’t have a a side suffix are assumed to belong to the center of the character. They are rigged with two Action constraints with influence 0.5 that are controlled by each of the control bones.

Frame Start & End

Specifies the frame range of the action that will be used by the created constraints.

Target Space, Transform Channel

Specifies the coordinate space and transformation channel of the target bone that should be used.

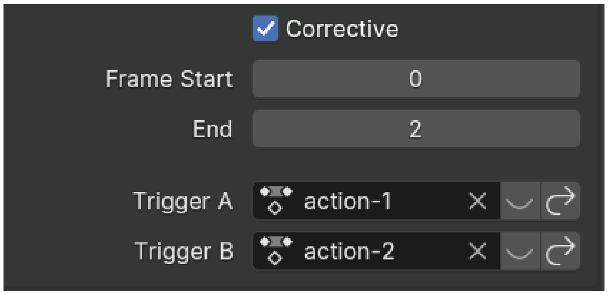
Min, Max

Specifies the range of the transformation channel values that is mapped to the specified action frame range.

Default Frame

Shows the frame within the action that maps to the neutral value (1 for scale and 0 otherwise) of the transformation channel, as computed from the specified range values.

Corrective Actions



☑ Corrective
Frame Start: 0
End: 2
Trigger A: action-1
Trigger B: action-2

Corrective actions are applied based on the progress of two other actions from the list, and are used to improve the pose when they are used together.

Frame Start & End

Specifies the frame range of the action that will be used by the created constraints.

Trigger A & B

Specifies the two actions that control the correction. The interface rows contain buttons to show the settings for that action, or jump to it in the list.

The progress of the corrective action from the start to the end frame is calculated as the product of the progress values of the two trigger actions. Thus, the start frame is applied when either of the triggers is at the start frame, and the end frame is used when both are at their end frame.

Corrective actions must be below their triggers in the list, which is enforced via an implicit reorder even if violated.

Tip

Corrective actions behave in the most intuitive way when both triggers have the Default Frame equal to Start Frame. To create a corrective action in such case:

- Create the two trigger actions, add them to the panel and generate the rig.
- Pose your controls so that both trigger actions are fully activated to the end frame.
- Pose and keyframe the necessary corrections in the end frame of the new action, while keying the start frame to the neutral values.
- Add the newly created action to the end of the list in the panel and configure its settings.

Rig Types

Rig types are components used by Rigify to process specific parts of the meta-rig when generating the armature. They represent common character features, like the spine, limbs, fingers etc.

Note

The list of available rig types appears in the Bone properties tab when the bone is selected in Pose Mode. Scroll down the Properties to find Rigify Type panel.

This documents rig types that are bundled with Rigify.

- Basic
    - basic.copy_chain
    - basic.pivot
    - basic.raw_copy
    - basic.super_copy
- Spines
    - spines.super_spine
    - spines.basic_spine
    - spines.basic_tail
    - spines.super_head
- Limbs
    - limbs.simple_tentacle
    - limbs.super_finger
    - limbs.super_limb
    - limbs.arm
    - limbs.leg
    - limbs.paw
    - limbs.front_paw
    - limbs.rear_paw
    - limbs.super_palm

  - limbs.spline_tentacle
- Faces
  - faces.super_face
- Skin
  - skin.basic_chain
  - skin.stretchy_chain
  - skin.anchor
  - skin.glue
  - skin.transform.basic
- Face
  - face.basic_tongue
  - face.skin_eye
  - face.skin_jaw

Basic

These rig types are used to generate simple single-bone features, and for custom rigging done directly in the meta-rig.

The single-bone rig types must be applied separately to every bone even within a connected chain, and can have connected children controlled by a different rig type. This is unlike chain-based rig types that usually consume the whole connected chain.

basic.copy_chain

Copies the bone chain keeping all the parent relations within the chain untouched. Useful as a utility rig type for custom rigs.

Requirement: A chain of at least two connected bones.

Control (Boolean)
When enabled control bones and widgets will be created.

Deform (Boolean)
When enabled deform bones will be created.

basic.pivot

Single-bone rig type that creates a ‘custom pivot’ control for rotating and scaling its child sub-rigs.

This type of control transforms its children when rotated or scaled, while moving it merely changes the pivot point used by rotation or scaling.

Master Control
When enabled an extra parent control bone with a box widget is created to allow moving the rig. It is also required by all other options besides

Deform Bone.

Widget Type

Allows selecting one of the predefined widgets to generate for the master control instead of the default cube.

Switchable Parent

Generates a mechanism for switching the effective parent of the rig based on the value of a custom property.

Register Parent

Registers the rig as a potential parent scope for its child sub-rigs' parent switches.

Tags

Specifies additional comma-separated tag keywords for the registered parent scope. They can be used by other rigs to filter parent choices, or for selecting the default parent.

Some of the existing tags that are useful here:

`injected` (special)

The parent scope will be made available for all children of the *parent* sub-rig, rather than just this rig's children.

`held_object`

A control for the object held in the character's hand. Preferred by finger IK.

The `injected,held_object` combination is perfect for such a control.

Pivot Control

Disabling this avoids generating the actual custom pivot control, effectively turning this rig type into a version of basic.super_copy with parent switching support and a different widget.

Deform Bone

When enabled a deform bone will be created.

basic.raw_copy

Single-bone rig type that copies the bone without the `ORG-` name prefix.

Normally all bones copied from the meta-rig are prefixed with `ORG-` and placed on an invisible layer. This precludes their use as controls or deforming bones, which makes it difficult to transfer complex fully custom rigging verbatim from the meta-rig.

This rig type does not add the automatic prefix, thus allowing an appropriate `ORG-`, `MCH-` or `DEF-` prefix to be manually included in the meta-rig bone name, or alternatively using no prefix to create a control bone.

Relink Constraints

Allows retargeting constraints belonging to the bone to point at bones created in the process of generating the rig, thus allowing custom rigging to integrate with generated bones.

To use this feature, add `@` and the intended target bone name to the constraint name, resulting in the `...@bone_name` syntax. After all bones of the rig are generated, the constraint target bone will be replaced. If the new bone name is just `CTRL`, `MCH` or `DEF`, this will just replace the `ORG` prefix in the existing target bone name. For the Armature constraint you can add a `@` suffix for each target, or just one `@CTRL`, `@MCH` or `@DEF` suffix to update all.

Parent

If the field is not empty, applies the same name substitution logic to the parent of the bone.

When this feature is enabled, the bone will not be automatically parented to the root bone even if it has no parent; enter `root` in the *Parent* field if that is necessary.

basic.super_copy

Single-bone rig type that simply copies the bone. Useful as utility rig type for adding custom features or specific deform bones to your rigs.

Control (Boolean)
When enabled a control bone and widget will be created.

Widget (Boolean)
When enabled a widget will be created in replacement to the standard.

Widget Type (String):
Allows selecting one of the predefined widget types to generate instead of the default circle.

Deform (Boolean)
When enabled a deform bone will be created.

Relink Constraints
Works the same as in the basic.raw_copy rig. In addition, when enabled any constraints that have names prefixed with `CTRL:` are moved to the control, and with `DEF:` to the deform bone.

Spines

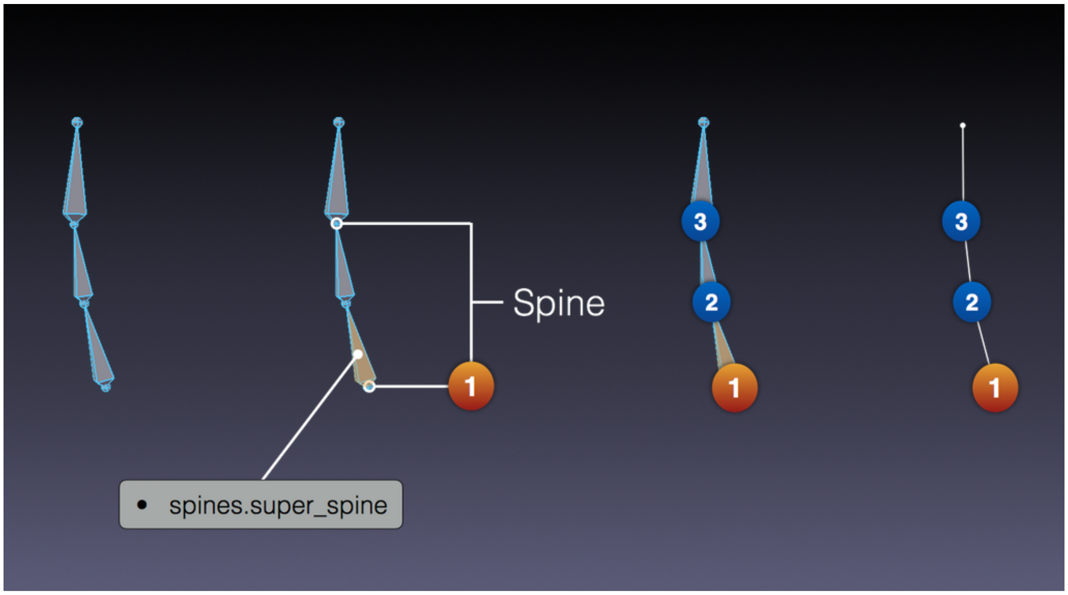
These rigs are used to generate spine structures, including the head and tail.

spines.super_spine

Will create a complete bendy and stretchy b-bones spine system based on bone numbers of your bone chain and user defined options.

This is a composite wrapper of spines.basic_spine, spines.super_head and spines.basic_tail. Note that for the tail, the direction of the bones is reversed compared to the separate rig.

Requirement: A chain of at least three connected bones (base system).



Spine required bones.

Pivot Position (integer)
: Defines the pivot position for torso and hips.

Head (Boolean)

When checked neck and head systems will be added to your spine rig.

Neck Position (integer)

Defines the bone where the neck system starts. The last bone will always be the head system. If neck position is the last bone of the chain, then only the head system will be created ignoring the neck.

Tail (Boolean)

When checked tail system will be added to your spine rig.

Tail Position (integer)

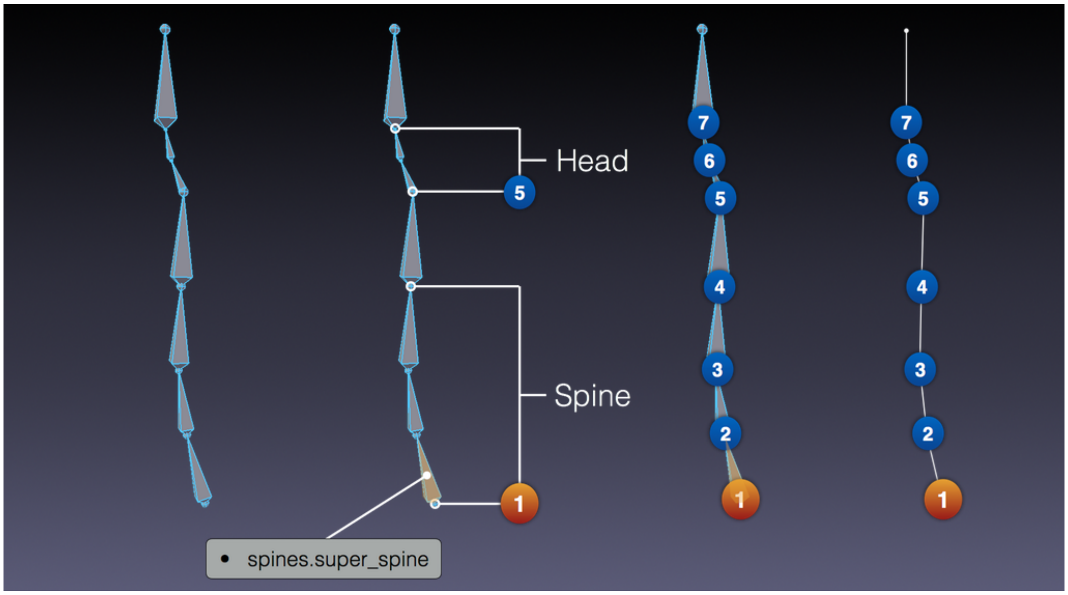
Defines the bone where the tail system starts. The next bone will always be the hips system.

X, Y, Z (Boolean)

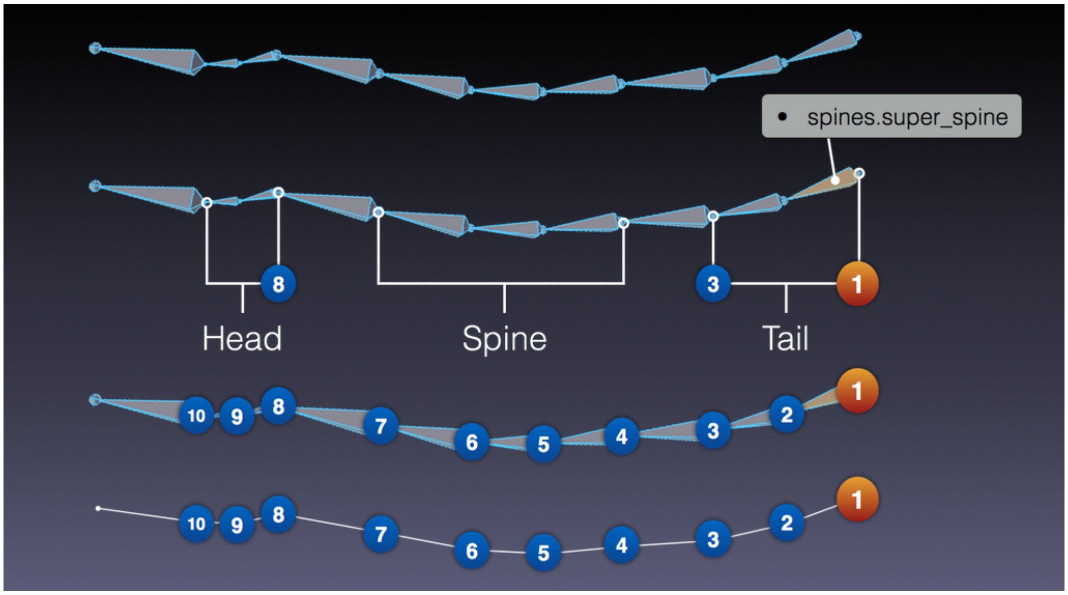
When generating a tail, specifies which local axis rotations should be replicated along the chain.

Assign Tweak Layers

If enabled, allows placing the Tweak controls in different bone collections from the IK bones.



Spine default bones.



Spine with tail bones.

spines.basic_spine

Defines a bendy and stretchy b-bones spine.

Pivot Position (integer)
    Defines the pivot position for torso and hips.

Assign Tweak Layers
    If enabled, allows placing the Tweak controls in different bone collections from the IK bones.

FK Controls
    Specifies whether to generate an FK control chain.

Assign FK Layers
    If enabled, allows placing the FK chain in different bone collections from the IK bones.

spines.basic_tail

Defines a bendy and stretchy b-bones tail.

X, Y, Z (Boolean)

Specifies which local axis rotations should be replicated along the chain from each control bone to the following one.

Assign Tweak Layers

If enabled, allows placing the Tweak controls in different bone collections from the IK bones.

spines.super_head

Defines a head rig with follow torso controls.

Assign Tweak Layers

If enabled, allows placing the Tweak controls in different bone collections from the IK bones.

Limbs

These rig types handle generation of different kind of limbs and their features, like fingers.

limbs.simple_tentacle

Will create a simple bendy and stretchy b-bones tentacle chain, which can optionally replicate local rotation from preceeding bones to the subsequent ones for use in cases like fingers.

Requirement: A chain of at least two connected bones.

Automation Axis (X, Y, Z, None)
: Enables the automation on the selected axis. Multiple axis or none can be selected holding `Shift-LMB`. When enabled the subsequent control bones will copy the local rotations from the previous ones. The option is accessible in the controls of the final rig as a Copy Rotation constraint and can be disabled even after rig is generated, or at animation time.

Assign Tweak Layers
: If enabled, allows placing the Tweak controls in different bone collections from the main controls.

limbs.super_finger

Will create a bendy and stretchy finger chain with a master control bone that controls the rotation of all joints through its scale.

Requirement: A chain of at least two connected bones.

Bend Rotation Axis (Automatic, X, Y, Z, -X, -Y, -Z)
: Defines the automatic rotation axis to be linked to the scale of the master bone.

B-Bone Segments (integer)

Defines the number of b-bone segments each tweak control will be split into.

IK Control

Generates a very simple IK mechanism with only one control.

IK starts its work with the shape of the finger defined by FK controls and adjusts it to make the fingertip touch the IK control. It is designed as a tool to temporarily keep the fingertip locked to a surface it touches, rather than a fully featured posing system.

To improve performance, the switchable parent for the IK control contains only one option beside None. Thus it is advised to add a ‘held object’ control using the basic.raw_copy rig to act as the common parent for the fingers with a fully functional parent switch.

IK Local Location

Specifies the value of the Local Location option for IK controls, which decides if the location channels are aligned to the local control orientation or world.

Assign Tweak Layers

If enabled, allows placing the Tweak controls in different bone collections from the main controls.

Assign Extra IK Layers

If enabled, allows placing the extra IK control in different bone collections from the main controls.

Note

Rotation Axis (Bend Rotation Axis in the case of limbs.super_finger) affects the roll of the generated bones. Automatic mode recalculates the generated bones roll while any of the Manual modes copy the roll of the meta-rig bones.

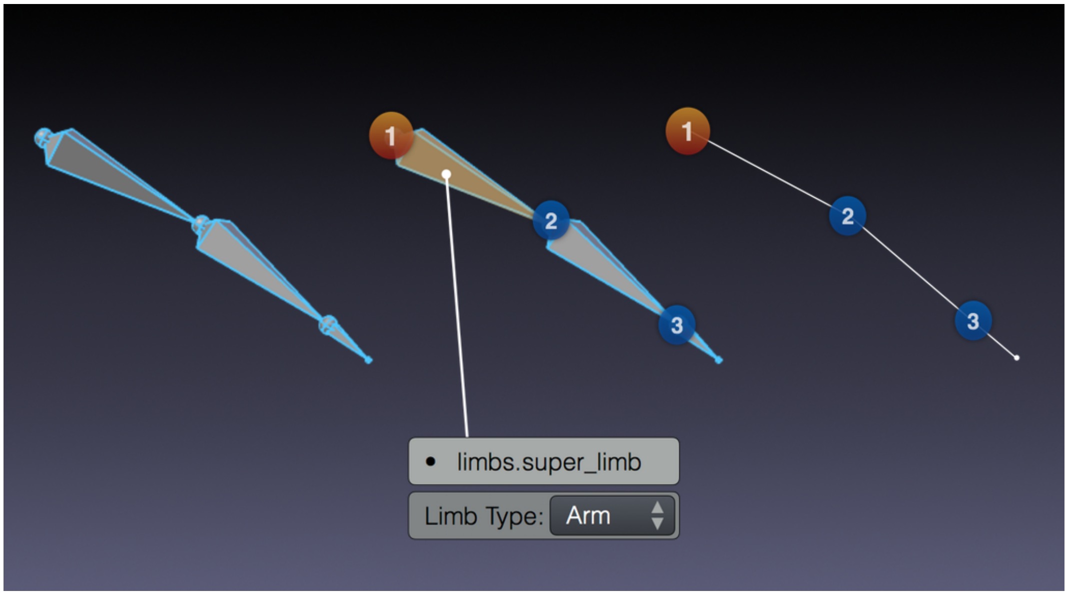
limbs.super_limb

A backwards compatibility wrapper around limbs.arm, limbs.leg and limbs.paw.

limbs.arm

Will create a fully featured bendy and stretchy arm depending on the user-defined options.

Requirement: A chain of three connected bones (upper_arm, forearm, hand).



- limbs.super_limb

Limb Type: Arm

Arm required bones.

IK Wrist Pivot

Generates an extra child of the hand IK control that rotates around the tail of the hand bone.

Rotation Axis (Automatic, X, Z)

Defines the bend axis for the IK chain. FK chains will have a totally free degree of rotation on all axes.

Limb Segments (integer)
: Defines the number of additional tweak controls each limb bone will have on the final rig.

B-Bone Segments (integer)
: Defines the number of b-bone segments each tweak control will be split into.

Custom IK Pivot
: Generates an extra control for the end of the IK limb that allows rotating it around an arbitrarily placed pivot.

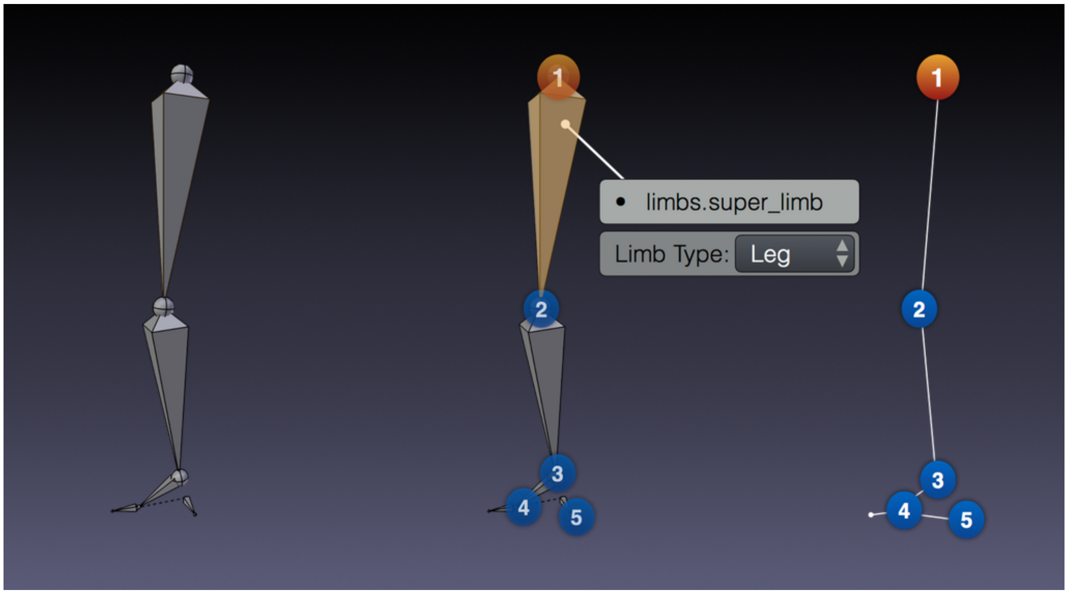
Assign FK Layers
: If enabled, allows placing the FK chain in different bone collections from the IK bones.

Assign Tweak Layers
: If enabled, allows placing the Tweak controls in different bone collections from the IK bones.

limbs.leg

Will create a fully featured bendy and stretchy leg depending on the user-defined options.

Requirement: A chain of four connected bones (thigh, shin, foot, toe) with one unconnected child of the foot to be used as the heel pivot.



Leg required bones.

Foot Pivot (Ankle, Toe, Ankle & Toe)

Specifies where to put the pivot location of the main IK control, or whether to generate an additional pivot control at the base of the toe.

Separate IK Toe

Specifies that two separate toe controls should be generated for IK and FK instead of sharing one bone. This is necessary to get fully correct IK-FK snapping in all possible poses.

Toe Tip Roll

Generates a slider to switch the heel control to pivot on the tip rather than the base of the toe (for roll this obviously only applies on forward roll).

Rotation Axis (Automatic, X, Z)

Defines the bend axis for the IK chain. FK chains will have a totally free degree of rotation on all axes.

Limb Segments (integer)

Defines the number of additional tweak controls each limb bone will have on the final rig.

B-Bone Segments (integer)

Defines the number of b-bone segments each tweak control will be split into.

Custom IK Pivot

Generates an extra control for the end of the IK limb that allows rotating it around an arbitrarily placed pivot.

Assign FK Layers

If enabled, allows placing the FK chain in different bone collections from the IK bones.

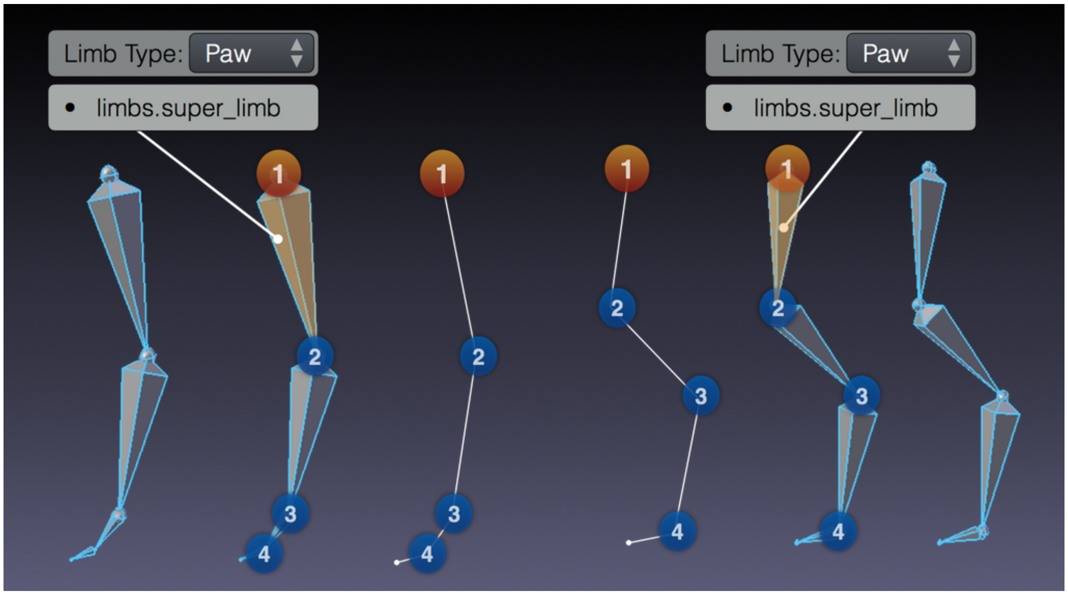
Assign Tweak Layers

If enabled, allows placing the Tweak controls in different bone collections from the IK bones.

limbs.paw

Will create a fully featured bendy and stretchy paw depending on the user-defined options.

Requirement: A chain of four or five connected bones (thigh, shin, paw, *optional* digit, toe).



Limb Type: Paw

limbs.super_limb

Limb Type: Paw

limbs.super_limb

Front/Rear paw required bones.

Rotation Axis (Automatic, X, Z)
Defines the bend axis for the IK chain. FK chains will have a totally free degree of rotation on all axes.

Limb Segments (integer)
Defines the number of additional tweak controls each limb bone will have on the final rig.

B-Bone Segments (integer)
Defines the number of b-bone segments each tweak control will be split into.

Custom IK Pivot
Generates an extra control for the end of the IK limb that allows rotating it around an arbitrarily placed pivot.

Assign FK Layers
If enabled, allows placing the FK chain in different bone collections from the IK bones.

Assign Tweak Layers
If enabled, allows placing the Tweak controls in different bone collections from the IK bones.

limbs.front_paw

Derivative of limbs.paw with extended IK suitable for use in front paws. The additional IK limits the degree of change in the angle between shin and paw bones (2nd and 3rd) as the main IK control moves and rotates.

For best results, the shin bone should not be parallel to either thigh or paw in rest pose, i.e. there should be some degree of bend in all joints of the paw.

Heel IK Influence

Influence of the extended IK. At full rotating the main IK control or digit bone would not affect the rotation of the paw bone, while lower values provide some blending.

limbs.rear_paw

Derivative of limbs.paw with extended IK suitable for use in rear paws. The additional IK tries to maintain thigh and paw bones (1st and 3rd) in a nearly parallel orientation as the main IK control moves and rotates.

For best results, thigh and paw bones should start nearly parallel in the rest pose.

limbs.super_palm

Will create a palm system based on the distance between palm bones.

Requirement: At least two bones child of the same parent. The property has to be set on the inner palm bones (think it as index's metacarpus), the rig control will appear on the last palm bone (think it as pinky's metacarpus).

Both Sides
Generates controls on both sides of the palm, with influence on inner bones blended between them.

Primary Rotation Axis (X, Z)
Defines the automatic rotation axis to be used on the palm bones.

limbs.spline_tentacle

This rig type implements a flexible tentacle with an IK system using the Spline IK constraint. The control bones define control points of a Bezier curve, and the bone chain follows the curve.

The curve control points are sorted into three groups: start, middle and end. The middle controls are always visible and active, while the other two types can be shown and hidden dynamically using properties; when enabled they appear next to the corresponding permanent start/end control and can be moved from there.

Extra Start Controls

Specifies the number of optional start controls to generate.

Middle Controls

Specifies the number of middle controls to generate.

Extra End Controls

Specifies the number of optional end controls to generate.

Tip Control:

Specifies how the curve stretching and the final control bone work:

Stretch To Fit

Stretches the whole bone chain to fit the length of the curve defined by the controls.

An end twist control is generated to control the twist along the chain.

Direct Tip Control

Generates an IK end control, which directly controls the final bone of the chain similar to how regular IK works for limbs, as well as controlling the end of the bezier curve. The middle bones of the chain stretch to follow the curve and cover the gap.

The rig automatically deduces twist of up to 180 degrees based on the orientation of the end control. Higher amounts of twist have to be dialled in through an End Twist Estimate slider to avoid flipping.

Manual Squash & Stretch

This mode allows full manual control over the chain scaling, while the chain covers as much of the curve as it can given its current length.

The start control of the chain manages its uniform squash & stretch scale, while the end twist control manages both the twist of the chain, as well as its scale at the tip (blended gradually along the length).

Radius Scaling

Allows scaling the controls to control the thickness of the chain through the curve.

Maximum Radius

Specifies the maximum scale allowed by the Radius Scaling feature.

FK Controls

Generates an FK control chain and IK-FK snapping.

Assign FK Layers

If enabled, allows placing the FK chain in different bone collections from the IK bones.

Faces

faces.super_face

Will create a face system based on the bones child to the parent that has the property set on it.

Requirement: All the face bones bundled in the `faces.super_face` sample had to be present and child of the master bone that has the Rigify-type *face* property set.

Note

This rig type is being deprecated in favor of a new modular skin and face rigging system.

Skin

These rigs implement a flexible system for rigging skin using multiple interacting B-Bone chains. This is developed as the base for a new modular Rigify face rig. These are the main ideas of the system:

Generic B-Bone Chain

> One core idea of the system is that most of the deformation should be implemented using a standard powerful B-Bone chain rig. These chains support advanced behavior by interacting with other rig components. This is in contrast to having multiple domain-specific rigs that each generate their own deform chains.
>
> The implementation provides two versions of the chain rig: skin.basic_chain merely attaches B-Bones to the controls with no automation added to the controls themselves. The skin.stretchy_chain rig in addition interpolates motion of the end (and an optional middle) controls to the other controls of the chain.

Automatic Control Merging

> The deformation part of the system consists of chains of one or more B-Bones connecting control points (nodes). Whenever controls for two chains would completely overlap, they are automatically merged.
>
> For each merged control, one of the chains is selected as the owner, based on heuristic factors like parent depth from root, presence of `.T`/`.B` `.L`/`.R` symmetry markers, and even alphabetical order as the last resort. This can be overridden by an explicit priority setting in cases when it guesses wrong.
>
> The owner and its parents determine additional automation that is placed on the control. As a special case, if a control is merged with its `.T`/`.B` `.L`/`.R` symmetry counterparts (detected purely by naming), the automation from all of the symmetry siblings of the owner is averaged.

Parent Controllers

> Rather than simply using the parent meta-rig bone (ORG) as parent for controls and chain mechanisms, the new system includes an interface for parent rigs. It explicitly provide parent bones and generate control parent automation mechanisms for their child chain controls by inheriting from the appropriate base and overriding methods.
>
> This allows implementing rigs that integrate and manage their child chains in intelligent ways in order to add extra automation specific to certain areas. The base skin system includes one simple example skin.transform.basic rig, which translates its child control points according to its control bone transformation.

Custom Rigging

> Finally, the new system provides ways to integrate with custom automation directly included in the meta-rig via two extra rig components.
>
> The skin.anchor rig generates a single control with inherited constraints etc., similar to basic.super_copy. However, it also integrates into the skin system as a zero length chain with highest priority. This allows overriding the normal behavior by providing a control point under full control of the user, which other chains would automatically attach to.
>
> The skin.glue rig on the other hand will attach itself to the control that is generated at its position (it is an error if there is none). It can be used to read the position of the control from custom rigging in the meta-rig, or inject constraints into the control bone. It is possible to also detect the control at the tail of the glue bone and use it as target in the constraints, thus copying transformation between the controls.

skin.basic_chain

This is the basic chain rig, which bridges controls with B-Bones but does not add any automation to the controls themselves.

When controls are merely moved, the chains behave as if using standard automatic handles, but rotating and optionally scaling the controls will adjust the result.

B-Bone Segments

Specifies the number of segments to use. Setting this to 1 disables all advanced behavior and merely bridges the points with a Stretch To bone.

Merge Parent Rotation and Scale

This can be enabled to let the chain respond to rotation and scale induced by parents of controls owned by other chains that this chain's control merged into.

Use Handle Scale

Enables using control scale to drive scale and/or easing of the B-Bone.

Connect With Mirror

Specifies whether the ends of the chain should smoothly connect when merging controls with its `.T`/`.B` `.L`/`.R` symmetry counterpart. The relevant option must be enabled on both chains to work.

Connect Matching Ends

Specifies whether the end of the chain should connect to the opposite end of a different chain when merging controls. Thus forming a continuous smooth chain in the same direction. The relevant options must be enabled on both chains.

Sharpen Corner

Specifies whether the rig should generate a mechanism to form a sharp corner at the relevant connected end, depending on the angle formed by adjacent control locations. When the control angle becomes sharper than the specified value, ease starts reducing from 1 to 0.

Orientation

Specifies that the controls should be oriented the same as the selected bone, rather than being aligned to the chain.

Copy To Selected

Copy to selected rigs that have the same option. Thus allowing to indiscriminately selecting bones without assigning unnecessary values.

Chain Priority

Allows overriding the heuristic used to select the primary owner when merging controls.

skin.stretchy_chain

This rig extends the basic chain with automation that propagates movement of the start and end, and an optional middle control, to other controls. This results in stretching the whole chain when moving one of the ends, rather than just the immediately adjacent B-Bones.

Middle Control Position

Specifies the position of the middle control within the chain; disabled when zero.

Falloff

Specifies the influence falloff curves of the start, middle and end controls. Zero results in linear falloff, increasing widens the influence, and -10 disables the influence propagation from that control completely.

Spherical Falloff

Toggle buttons to change the shape of the falloff curve from a power curve that at falloff 1 forms a parabola $1 - x^{2^f}$ to a curve forming a circle $(1 - x^{2^f})^{2^{-f}}$.

Falloff Along Chain Curve

Computes the falloff curve along the length of the chain, instead of projecting on the straight line connecting its start and end points.

Propagate Twist

Specifies whether twist of the chain should be propagated to control points between main controls.

Propagate Scale
Specifies whether perpendicular scaling of the chain should be propagated to control points between main controls.

Propagate to Controls
Allows other chains to see propagated twist and scale via *Merge Parent Rotation and Scale* when their controls are merged into this chain, instead of it being completely local to this chain.

Primary Control Layers
Optionally specifies bone collections for the end controls.

Secondary Control Layers
Optionally specifies bone collections for the middle control, falling back to *Primary Control Layers* if not set.

The main controls with active falloff have the effect of *Merge Parent Rotation and Scale* automatically enabled just for them.

skin.anchor

This rig effectively acts as a zero-length chain with highest priority, ensuring that it becomes the owner when merging controls with other chains. And also allowing one to input custom automation influence into the skin system.

All constraints on the meta-rig bone are moved to the created control.

Generate Deform Bone
Creates a deformation bone parented to the control.

Suppress Control
Makes the control a hidden mechanism bone to hide it from the user.

Widget Type
Selects which widget to generate for the control.

Relink Constraints

Operates the same as in basic.raw_copy, except all constraints are moved from ORG to the control bone.

Orientation
: Specifies the bone used to orient the control, like for other chains.

skin.glue

This rig is in concept similar to skin.anchor, but instead of overriding controls, it is used to read or adjust the state of controls generated by other rigs. The head of the bone must overlap a control of another skin rig.

The rig sets up its ORG bone to read the state of the control, while moving all constraints that were originally on the bone to the control.

Glue Mode

Specifies how the ORG bone is connected to the skin control.

Child Of Control

Makes the ORG bone a child of the control bone.

Mirror Of Control

Makes the ORG bone a sibling of the control with a Copy Transforms constraint from the control. The resulting local space transformation is the same as control's local space.

Mirror With Parents

Parents the ORG bone to the parent automation a control owned by the glue rig would have had, while making it follow the actual control. This includes both direct and parent-induced motion of the control into the local space transformation of the bone.

Deformation Bridge

Other than adding glue constraints to the control, the rig acts as a one segment basic deform chain. This is convenient when a pair of controls need to be bridged both with glue and a deform bone.

Relink Constraints

Operates the same as in basic.raw_copy, except all constraints are moved from ORG to the control bone.

Use Tail Target

Relinks `TARGET` or any constraints with an empty target bone and no relink specification to reference the control located at the tail of the glue bone.

Target Local With Parents

Switches the tail target to operate similarly to *Mirror With Parents*.

Add Constraint

Allows to add a typical glue constraints with specific *Influence*, as if it were at the start of the ORG bone constraint stack.

skin.transform.basic

This rig provides a simplistic parent controller, which uses regular translation, rotation, or scale to modify locations but not orientations or scale of its child chain controls.

Generate Control

Specifies whether to generate a visible control, or use the transformation of the ORG bone as a part of more complex and specific rig setup.

Face

These rig types implement components of a modular face.

face.basic_tongue

Generates a simple tongue, extracted from the original PitchiPoy super_face rig.

B-Bone Segments (integer)
: Defines the number of b-bone segments each tweak control will be split into.

Primary Control Layers
: Optionally specifies bone collections for the main control.

face.skin_eye

Implements a skin system parent controller that manages two skin chains for the top and bottom eyelids in addition to generating the eye rotation mechanism.

The rig must have two child skin chains with names tagged with `.T` and `.B` symmetry to mark the top and bottom eyelid, which are connected at their ends forming eye corners. The chains are rigged to follow the surface of the eye and twist to its normal.

In addition, it creates target controls for aiming the eye, including a master control shared by all eyes under the same parent rig. The eyelids are rigged to follow the movement of the eyeball with adjustable influence.

Eyeball and Iris Deforms
: Generates deform bones for the eyeball and the iris, the latter copying XZ scale from the eye target control. The iris is located at the tail of the

ORG bone.

Eyelid Detach Option
: Generates a slider to disable the mechanism that keeps eyelid controls stuck to the surface of the eye.

Split Eyelid Follow Slider
: Generates two separate sliders for controlling the influence of the eye rotation on X and Z eyelid motion.

Eyelids Follow Default
: Depending on *Split Eyelid Follow Slider*, specifies the default values for the split follow sliders, or fixed factors to be multiplied with the single common follow influence slider value.

face.skin_jaw

Implements a skin system parent controller that manages one or more loops of mouth skin chains in response to the movement of jaw and mouth controls.

The rig must have one or more child chain loops, each formed by four skin chains tagged with `.T`/`.B` and `.L`/`.R` symmetrical names.

The lip loops are sorted into layers based on the distance from corners to the common center and rigged with blended influence of the jaw and the master mouth control. Other child rigs become children of the jaw.

Bottom Lip Influence
: Specifies the influence of the jaw on the inner bottom lip with mouth lock disabled.

Locked Influence
: Specifies the influence of the jaw on both lips of locked mouth.

Secondary Influence Falloff
: Specifies the factor by which influence fades away with each successive lip loop (for bottom lip loops the blend moves away from inner bottom

lip to full jaw influence).

Feature Sets

Rigify allows third party developers to implement sub-addons, called *Feature Sets*, which can provide new Meta-Rigs and Rig Types. Similar to regular add-ons, they can be installed from zip-files through Rigify settings.

These are some examples of *Feature Sets* currently provided by past and current Rigify developers:

Cessen’s Rigify Extensions

This feature set provides the original Rigify rigs by Nathan Vegdahl, minimally ported and repackaged to work without switching Rigify to legacy mode. Note that their names were changed, so meta-rigs designed for legacy mode aren’t directly compatible.

Experimental Rigs by Alexander Gavrilov

Rig experiments, some of which might be included in Rigify in the future. Examples include limbs with an extra IK system based at knee/elbow, a spline based tentacle, and more.

You can install these packages by clicking *Code* ▸ *Download ZIP*, and then install the downloaded file through Rigify settings.

Developer documentation is available on the Blender Developer Documentation.

System

Important

Work In Progress

These add-ons relate to showing information about objects and scenes.

- Manage UI Translations

Manage UI Translations

Todo

Enabling Add-on

1. Open Blender and go to Add-ons section of the Preferences.
2. Search “Manage UI Translations” and check the *Enable Extension* checkbox.

Description

See Blender translation guide in the Developer Handbook.

Reference

Category:: System
Description:: Allows managing UI translations directly from within Blender (update main po-files, update scripts’ translations, etc.).
Location:: *Topbar* ▸ *File menu*, Text editor, any UI control
File:: ui_translate folder
Author:: Bastien Montagne
Note:: This add-on is bundled with Blender.

Advanced

This chapter covers advanced use (topics which may not be required for typical usage).

- Using Blender From The Command Line
- Scripting & Extending Blender
- Creating Extensions
- Application Templates
- Keymap Customization
- Working Limits
- Operators
- Blender’s Directory Layout
- Deploying Blender in Production
- Appendices

Using Blender From The Command Line

The *Console Window*, also called a *Terminal*, is an operating system text window that displays messages about Blender's operations, status, and internal errors.

When Blender is manually started from a terminal, Blender output is shown in the *Console Window* it is started from.

Use Cases:

- For rendering animation.
- For automation and batch processing which require launching Blender with different arguments.
- For Python development, to see the output of the `print()` function.
- If Blender exits unexpectedly, the messages may indicate the cause or error.
- When troubleshooting, to see the output of `--debug` messages.

See: Launching from the Command Line for specific instructions on launching Blender from the command line.

Tip

Closing the *Console Window* will also close Blender, losing any unsaved work.

Launching from the Command Line

- Linux
- macOS

- Windows

Arguments

- Render Options
- Cycles Render Options
- Format Options
- Animation Playback Options
- Window Options
- Python Options
- Network Options
- Logging Options
- Debug Options
- GPU Options
- Misc Options

Sub-Commands

- Extensions Command Line Arguments
    - Package Management
    - Repository Management
    - Extension Creation

Workflows

- Rendering From The Command Line
    - Single Image
    - Animation
    - Cycles

Launching from the Command Line

- Linux
  - Quick Start
  - Details
- macOS
  - Quick Start
  - Details
- Windows
  - Quick Start
  - Details

Linux

Quick Start

Open a terminal, then go to the directory where Blender is installed, and run Blender like this:

```
cd <blender installation directory>
./blender
```

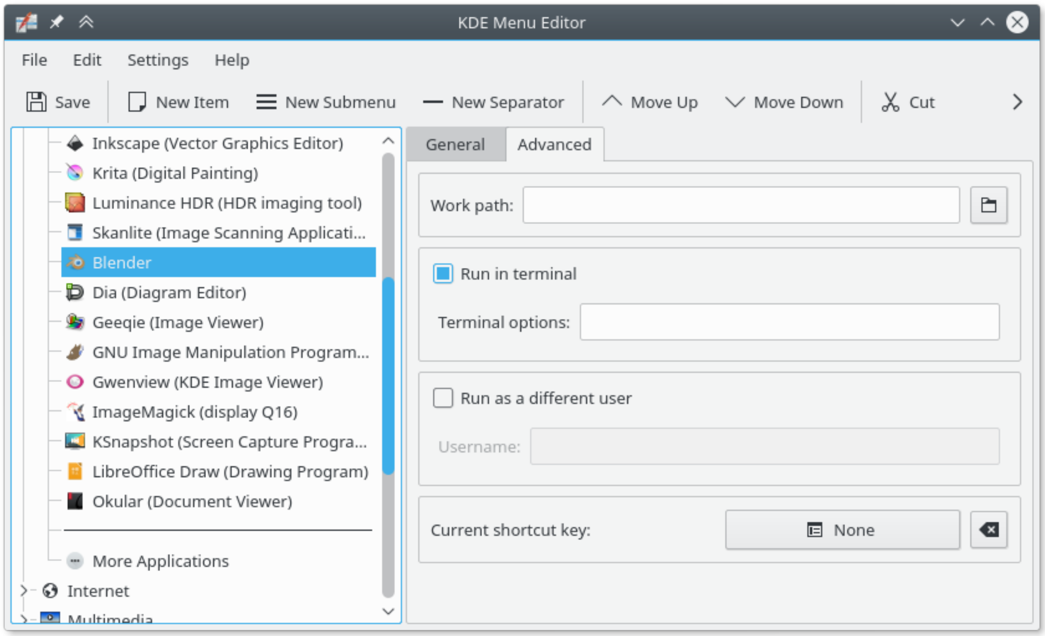
If you have Blender installed in your `PATH` (usually when Blender is installed through a distribution package), you can simply run:

```
blender
```

Details

Depending on your desktop environment setup, a Blender icon may appear on your desktop or an entry for Blender is added to your menu after you install Blender.

Many desktop environments support the ability to “Run in terminal”.


KDE Menu Editor
File
Edit
Settings
Help
Save
New Item
New Submenu
New Separator
Move Up
Move Down
Cut
Inkscape (Vector Graphics Editor)
Krita (Digital Painting)
Luminance HDR (HDR imaging tool)
Skanlite (Image Scanning Applicati...
Blender
Dia (Diagram Editor)
Geeqie (Image Viewer)
GNU Image Manipulation Program...
Gwenview (KDE Image Viewer)
ImageMagick (display Q16)
KSnapshot (Screen Capture Progra...
LibreOffice Draw (Drawing Program)
Okular (Document Viewer)
More Applications
Internet
Multimedia
General
Advanced

Work path:

☑ Run in terminal

Terminal options:

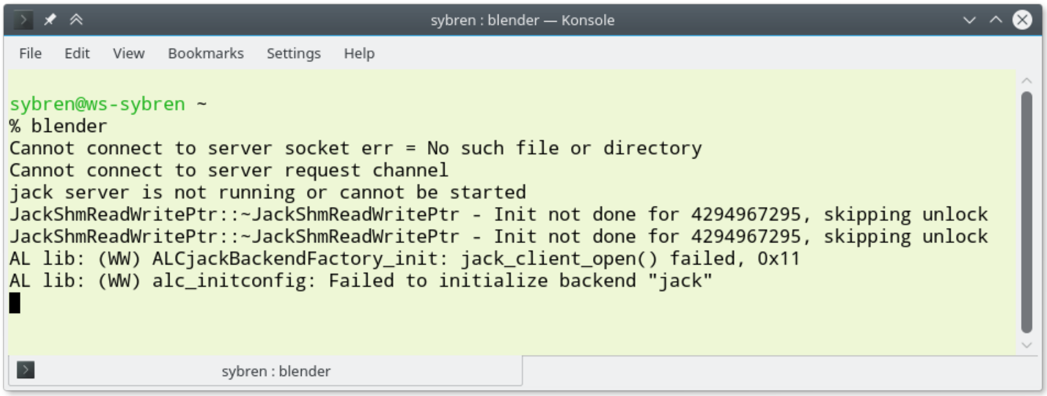
☐ Run as a different user

Username:

Current shortcut key: None

Configuring the KDE menu icon to start Blender from a terminal.

This screenshot shows Blender started from a Linux Terminal and the resulting text output written to it:


sybren : blender — Konsole
File
Edit
View
Bookmarks
Settings
Help
sybren : blender

```
sybren@ws-sybren ~
% blender
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for 4294967295, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for 4294967295, skipping unlock
AL lib: (WW) ALCjackBackendFactory_init: jack_client_open() failed, 0x11
AL lib: (WW) alc_initconfig: Failed to initialize backend "jack"
█
```

Starting Blender from a Linux terminal window.

macOS

Quick Start

Open the terminal application, and run the executable within the app bundle, with commands like this:

```
cd /Applications/Blender.app/Contents/MacOS
./Blender
```

If you need to do this often, you can add this directory to your `PATH`.

For that you can run the following procedure:

1. Open up a Terminal.
2. Run the following command: `sudo nano /etc/paths`.
3. Enter your password, when prompted.
4. Go to the bottom of the file, and enter `/Applications/Blender.app/Contents/MacOS`.
5. Enter `Ctrl-X` to quit.
6. Enter `Y` to save the modified buffer.

If you then open a new terminal, the following command will work:

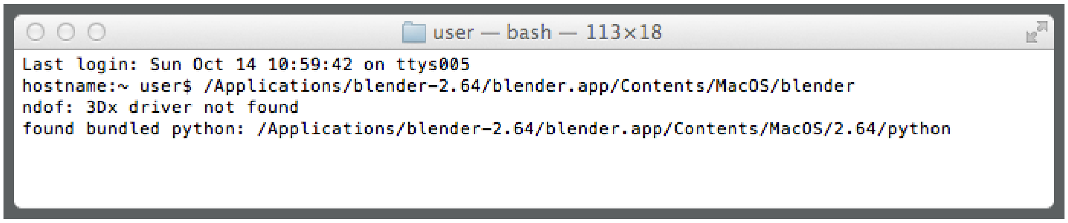
```
Blender
```

Details

macOS uses “files” with the `.app` extension called *applications*. These files are actually folders that appear as files in Finder. In order to run Blender you will have to specify that path to the Blender executable inside this folder, to get all output printed to the terminal. You can start a terminal from *Applications* ‣ *Utilities*. The path to the executable in the `.app` folder is `./Blender.app/Contents/MacOS/Blender`.

If you have Blender installed in the Applications folder, the following command can be used:

```
/Applications/Blender.app/Contents/MacOS/Blender
```



```
Last login: Sun Oct 14 10:59:42 on ttys005
hostname:~ user$ /Applications/blender-2.64/blender.app/Contents/MacOS/blender
ndof: 3Dx driver not found
found bundled python: /Applications/blender-2.64/blender.app/Contents/MacOS/2.64/python
```

Starting Blender from a macOS console window.

Windows

Quick Start

Open the Command Prompt, go to the directory where Blender is installed, and then run Blender:

```
cd c:\<blender installation directory>
blender
```

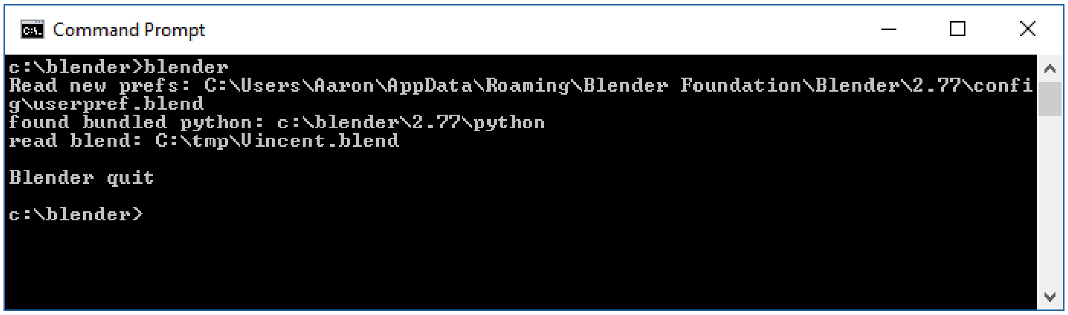
You can also add the Blender folder to your system `PATH` so that you do not have to `cd` to it each time.

Details

When Blender is started on a Microsoft Windows operating system, the *Console Window* (called the Command Prompt) is first created as a separate window on the desktop. The main Blender window will also appear and the *Console Window* will then be toggled off. To display the console again, go to *Window ‣ Toggle System Console*.

To start Blender from the command line you need to open an instance of Command Prompt. To do this, type `OSKey-R` then type `cmd`; this will open the Command Prompt window. You then need to find the path to the Blender executable. If you installed Blender via the installer then it can be found here:

```
C:\Program Files\Blender Foundation\Blender\blender.exe
```



```
c:\blender>blender
Read new prefs: C:\Users\Aaron\AppData\Roaming\Blender Foundation\Blender\2.77\confi
g\userpref.blend
found bundled python: c:\blender\2.77\python
read blend: C:\tmp\Vincent.blend

Blender quit

c:\blender>
```

Blender’s Console Window on Microsoft Windows.

The screenshot shows the Blender *Console Window* on Microsoft Windows directly after starting Blender and then a short while later after opening a file along with the relevant messages.

Command Line Arguments

Blender 4.3
Usage: `blender [args ...] [file] [args ...]`

Render Options

`-b`, `--background`

Run in background (often used for UI-less rendering).

The audio device is disabled in background-mode by default and can be re-enabled by passing in `-setaudo Default` afterwards.

`-a`, `--render-anim`

Render frames from start to end (inclusive).

`-S`, `--scene` `<name>`

Set the active scene `<name>` for rendering.

`-f`, `--render-frame` `<frame>`

Render frame `<frame>` and save it.

- `+<frame>` start frame relative, `-<frame>` end frame relative.
- A comma separated list of frames can also be used (no spaces).
- A range of frames can be expressed using `..` separator between the first and last frames (inclusive).

`-s`, `--frame-start` `<frame>`

Set start to frame `<frame>`, supports +/- for relative frames too.

`-e`, `--frame-end` `<frame>`

Set end to frame `<frame>`, supports +/- for relative frames too.

`-j`, `--frame-jump` `<frames>`

Set number of frames to step forward after each rendered frame.

`-o`, `--render-output` `<path>`

Set the render path and file name. Use `//` at the start of the path to render relative to the blend-file.

The `#` characters are replaced by the frame number, and used to define zero padding.

- `animation_##_test.png` becomes `animation_01_test.png`
- `test-######.png` becomes `test-000001.png`

When the filename does not contain `#`, the suffix `####` is added to the filename.

The frame number will be added at the end of the filename, eg:

```
blender -b animation.blend -o //render_ -F PNG -x 1 -a
```

`//render_` becomes `//render_####`, writing frames as `//render_0001.png`

`-E`, `--engine` `<engine>`

Specify the render engine. Use `-E help` to list available engines.

`-t`, `--threads` `<threads>`

Use amount of `<threads>` for rendering and other operations [1-1024], 0 to use the systems processor count.

Cycles Render Options

Cycles add-on options must be specified following a double dash.

`--cycles-device` `<device>`

Set the device used for rendering. Valid options are: `CPU` `CUDA` `OPTIX` `HIP` `ONEAPI` `METAL`.

Append +CPU to a GPU device to render on both CPU and GPU.

Example:

```
blender -b file.blend -f 20 -- --cycles-device OPTIX
```

`--cycles-print-stats`
Log statistics about render memory and time usage.

Format Options

`-F`, `--render-format` `<format>`
Set the render format. Valid options are: `TGA` `RAWTGA` `JPEG` `IRIS` `AVIRAW` `AVIJPEG` `PNG` `BMP` `HDR` `TIFF`.

Formats that can be compiled into Blender, not available on all systems: `OPEN_EXR` `OPEN_EXR_MULTILAYER` `FFMPEG` `CINEON` `DPX` `JP2` `WEBP`.

`-x`, `--use-extension` `<bool>`
Set option to add the file extension to the end of the file.

Animation Playback Options

`-a` `<options>` `<file(s)>`
Instead of showing Blender's user interface, this runs Blender as an animation player, to view movies and image sequences rendered in Blender (ignored if `-b` is set).

Playback Arguments:

`-p` `<sx>` `<sy>`
Open with lower left corner at `<sx>`, `<sy>`.

`-m`
Read from disk (Do not buffer).

`-f` `<fps>` `<fps_base>`
Specify FPS to start with.

`-j` `<frame>`
Set frame step to `<frame>`.

`-s` `<frame>`
Play from `<frame>`.

`-e` `<frame>`
Play until `<frame>`.

`-c` `<cache_memory>`
Amount of memory in megabytes to allow for caching images during playback. Zero disables (clamping to a fixed number of frames instead).

Window Options

`-w`, `--window-border`
Force opening with borders.

`-W`, `--window-fullscreen`
Force opening in full-screen mode.

`-p`, `--window-geometry` `<sx>` `<sy>` `<w>` `<h>`
Open with lower left corner at `<sx>`, `<sy>` and width and height as `<w>`, `<h>`.

`-M`, `--window-maximized`
Force opening maximized.

`-con`, `--start-console`
Start with the console window open (ignored if `-b` is set), (Windows only).

`--no-native-pixels`
Do not use native pixel size, for high resolution displays (MacBook `Retina`).

`--no-window-focus`
Open behind other windows and without taking focus.

Python Options

`-y`, `--enable-autoexec`
Enable automatic Python script execution.

`-Y`, `--disable-autoexec`
Disable automatic Python script execution (Python-drivers & startup scripts), (default).

`-P`, `--python` `<filepath>`
Run the given Python script file.

`--python-text` `<name>`
Run the given Python script text block.

`--python-expr` `<expression>`
Run the given expression as a Python script.

`--python-console`
Run Blender with an interactive console.

`--python-exit-code` `<code>`
Set the exit-code in [0..255] to exit if a Python exception is raised (only for scripts executed from the command line), zero disables.

`--python-use-system-env`
Allow Python to use system environment variables such as `PYTHONPATH` and the user site-packages directory.

`--addons` `<addon(s)>`
Comma separated list (no spaces) of add-ons to enable in addition to any default add-ons.

Network Options

`--online-mode`
Allow internet access, overriding the preference.

`--offline-mode`
Disallow internet access, overriding the preference.

Logging Options

`--log <match>`
Enable logging categories, taking a single comma separated argument. Multiple categories can be matched using a `.*` suffix, so `--log "wm.*"` logs every kind of window-manager message. Sub-string can be matched using a `*` prefix and suffix, so `--log "*undo*"` logs every kind of undo-related message. Use “^” prefix to ignore, so `--log "*,^wm.operator.*"` logs all except for `wm.operators.*` Use “*” to log everything.

`--log-level <level>`
Set the logging verbosity level (higher for more details) defaults to 1, use -1 to log all levels.

`--log-show-basename`
Only show file name in output (not the leading path).

`--log-show-backtrace`
Show a back trace for each log message (debug builds only).

`--log-show-timestamp`
Show a timestamp for each log message in seconds since start.

`--log-file <filepath>`
Set a file to output the log to.

Debug Options

`-d`, `--debug`
Turn debugging on.

- Enables memory error detection
- Disables mouse grab (to interact with a debugger in some cases)

- Keeps Python's `sys.stdin` rather than setting it to None

`--debug-value <value>`
Set debug value of `<value>` on startup.

`--debug-events`
Enable debug messages for the event system.

`--debug-ffmpeg`
Enable debug messages from FFmpeg library.

`--debug-handlers`
Enable debug messages for event handling.

`--debug-libmv`
Enable debug messages from libmv library.

`--debug-cycles`
Enable debug messages from Cycles.

`--debug-memory`
Enable fully guarded memory allocation and debugging.

`--debug-jobs`
Enable time profiling for background jobs.

`--debug-python`
Enable debug messages for Python.

`--debug-depsgraph`
Enable all debug messages from dependency graph.

`--debug-depsgraph-eval`
Enable debug messages from dependency graph related on evaluation.

`--debug-depsgraph-build`
Enable debug messages from dependency graph related on graph construction.

`--debug-depsgraph-tag`

Enable debug messages from dependency graph related on tagging.

`--debug-depsgraph-no-threads`
Switch dependency graph to a single threaded evaluation.

`--debug-depsgraph-time`
Enable debug messages from dependency graph related on timing.

`--debug-depsgraph-pretty`
Enable colors for dependency graph debug messages.

`--debug-depsgraph-uid`
Verify validness of session-wide identifiers assigned to ID data-blocks.

`--debug-ghost`
Enable debug messages for Ghost (Linux only).

`--debug-wintab`
Enable debug messages for Wintab.

`--debug-gpu`
Enable GPU debug context and information for OpenGL 4.3+.

`--debug-gpu-force-workarounds`
Enable workarounds for typical GPU issues and disable all GPU extensions.

`--debug-gpu-compile-shaders`
Compile all statically defined shaders to test platform compatibility.

`--debug-gpu-scope-capture`
Capture the GPU commands issued inside the give scope name.

`--debug-gpu-renderdoc`
Enable Renderdoc integration for GPU frame grabbing and debugging.

`--debug-wm`
Enable debug messages for the window manager, shows all operators in search, shows keymap errors.

`--debug-xr`
Enable debug messages for virtual reality contexts. Enables the OpenXR API validation layer, (OpenXR) debug messages and general information prints.

`--debug-xr-time`
Enable debug messages for virtual reality frame rendering times.

`--debug-all`
Enable all debug messages.

`--debug-io`
Enable debug messages for I/O (Collada, …).

`--debug-fpe`
Enable floating-point exceptions.

`--debug-exit-on-error`
Immediately exit when internal errors are detected.

`--debug-freestyle`
Enable debug messages for Freestyle.

`--disable-crash-handler`
Disable the crash handler.

`--disable-abort-handler`
Disable the abort handler.

`--verbose <verbose>`
Set the logging verbosity level for debug messages that support it.

`-q`, `--quiet`
Suppress status printing (warnings & errors are still printed).

GPU Options

`--gpu-backend`

Force to use a specific GPU backend. Valid options: `vulkan` (experimental), `metal`, `opengl`.

`--gpu-compilation-subprocesses`
Override the Max Compilation Subprocesses setting (OpenGL only).

Misc Options

`--open-last`
Open the most recently opened blend file, instead of the default startup file.

`--app-template <template>`
Set the application template (matching the directory name), use `default` for none.

`--factory-startup`
Skip reading the `startup.blend` in the users home directory.

`--enable-event-simulate`
Enable event simulation testing feature `bpy.types.Window.event_simulate`.

`--env-system-datafiles`
Set the `BLENDER_SYSTEM_DATAFILES` environment variable.

`--env-system-scripts`
Set the `BLENDER_SYSTEM_SCRIPTS` environment variable.

`--env-system-extensions`
Set the `BLENDER_SYSTEM_EXTENSIONS` environment variable.

`--env-system-python`
Set the `BLENDER_SYSTEM_PYTHON` environment variable.

`-noaudio`
Force sound system to None.

`-setaudio`

Force sound system to a specific device. `None` `Default` `SDL` `OpenAL` `CoreAudio` `JACK` `PulseAudio` `WASAPI`.

`-c`, `--command` `<command>`

Run a command which consumes all remaining arguments. Use `-c help` to list all other commands. Pass `--help` after the command to see its help text.

This implies `--background` mode.

`-h`, `--help`

Print this help text and exit.

`/?`

Print this help text and exit (Windows only).

`-r`, `--register`

Register blend-file extension for current user, then exit (Windows & Linux only).

`--register-allusers`

Register blend-file extension for all users, then exit (Windows & Linux only).

`--unregister`

Unregister blend-file extension for current user, then exit (Windows & Linux only).

`--unregister-allusers`

Unregister blend-file extension for all users, then exit (Windows & Linux only).

`-v`, `--version`

Print Blender version and exit.

`--`

End option processing, following arguments passed unchanged. Access via Python's `sys.argv`.

Argument Parsing

Arguments must be separated by white space, eg:

```
blender -ba test.blend
```

…will exit since `-ba` is an unknown argument.

Argument Order

Arguments are executed in the order they are given. eg:

```
blender --background test.blend --render-frame 1 --render-
output "/tmp"
```

…will not render to `/tmp` because `--render-frame 1` renders before the output path is set.

```
blender --background --render-output /tmp test.blend --render-
frame 1
```

…will not render to `/tmp` because loading the blend-file overwrites the render output that was set.

```
blender --background test.blend --render-output /tmp --render-
frame 1
```

…works as expected.

Environment Variables

BLENDER_USER_RESOURCES::
Replace default directory of all user files. Other `BLENDER_USER_*` variables override when set.

BLENDER_USER_CONFIG::	Directory for user configuration files.
BLENDER_USER_SCRIPTS::	Directory for user scripts.
BLENDER_USER_EXTENSIONS::	Directory for user extensions.
BLENDER_USER_DATAFILES::	Directory for user data files (icons, translations, ..).
BLENDER_SYSTEM_RESOURCES::	Replace default directory of all bundled resource files.
BLENDER_SYSTEM_SCRIPTS::	Directory to add more bundled scripts.
BLENDER_SYSTEM_EXTENSIONS::	Directory for system extensions repository.
BLENDER_SYSTEM_DATAFILES::	Directory to replace bundled datafiles.
BLENDER_SYSTEM_PYTHON::	Directory to replace bundled Python libraries.
OCIO::	Path to override the OpenColorIO configuration file.
TEMP::	Store temporary files here (MS-Windows).
TMPDIR::	Store temporary files here (UNIX Systems). The path must reference an existing directory or it will be ignored.

Extensions Command Line Arguments

Command for managing Blender extensions.

options:

`-h, --help`	show this help message and exit

subcommands:

Package Management

list::	List all packages.
sync::	Synchronize with remote repositories.
update::	Upgrade any outdated packages.
install::	Install packages.
install-file::	Install package from file.
remove::	Remove packages.

Repository Management

repo-list::	List repositories.
repo-add::	Add repository.
repo-remove::	Remove repository.

Extension Creation

build::	Build a package.
validate::	Validate a package.
server-generate::	Create a listing from all packages.

Package Management

Subcommand: `list`

usage:

```
blender --command extension list [-h] [-s]
```

List packages from all enabled repositories.

options:

`-h, --help`	show this help message and exit
`-s, --sync`	Sync the remote directory before performing the action.

Subcommand: `sync`

usage:

```
blender --command extension sync [-h]
```

Download package information for remote repositories.

options:

`-h, --help`	show this help message and exit

Subcommand: `update`

usage:

```
blender --command extension update [-h] [-s]
```

Download and update any outdated packages.

options:

`-h, --help`	show this help message and exit
`-s, --sync`	Sync the remote directory before performing the action.

Subcommand: `install`

usage:

```
blender --command extension install [-h] [-s] [-e] [--no-prefs]
                                    PACKAGES
```

positional arguments:

PACKAGES::	The packages to operate on (separated by `,` without spaces).

options:

`-h, --help`	show this help message and exit
`-s, --sync`	Sync the remote directory before performing the action.
`-e, --enable`	Enable the extension after installation.
`--no-prefs`	Treat the user-preferences as read-only, preventing updates for operations that would otherwise modify them. This means removing extensions or repositories for example, wont update the user-preferences.

Subcommand: `install-file`

usage:

```
blender --command extension install-file [-h] -r REPO [-e] [--
no-prefs]
                                         FILE
```

Install a package file into a user repository.

positional arguments:

FILE::	The packages file.

options:

`-h, --help`	show this help message and exit
`-r` *`REPO`*`, --repo` *`REPO`*	The repository identifier.
`-e, --enable`	Enable the extension after installation.

`--no-prefs`	Treat the user-preferences as read-only, preventing updates for operations that would otherwise modify them. This means removing extensions or repositories for example, wont update the user-preferences.

Subcommand: `remove`

usage:

```
blender --command extension remove [-h] [--no-prefs] PACKAGES
```

Disable & remove package(s).

positional arguments:

PACKAGES::	The packages to operate on (separated by `,` without spaces).

options:

`-h, --help`	show this help message and exit
`--no-prefs`	Treat the user-preferences as read-only, preventing updates for operations that would otherwise modify them. This means removing extensions or repositories for example, wont update the user-preferences.

Repository Management

Subcommand: `repo-list`

usage:

```
blender --command extension repo-list [-h]
```

List all repositories stored in Blender's preferences.

options:

`-h, --help` show this help message and exit

Subcommand: `repo-add`

usage:

```
blender --command extension repo-add [-h] [--name NAME]
                                     [--directory DIRECTORY]
                                     [--url URL]
                                     [--access-token
ACCESS_TOKEN]
                                     [--source SOURCE]
                                     [--cache BOOLEAN] [--
clear-all]
                                     [--no-prefs]
                                     ID
```

Add a new local or remote repository.

positional arguments:

ID::	The repository identifier.

options:

`-h, --help`	show this help message and exit
`--name` *`NAME`*	The name to display in the interface (optional).
`--directory` *`DIRECTORY`*	The directory where the repository stores local files (optional). When omitted a directory in the users directory is automatically selected.
`--url` *`URL`*	The URL, for remote repositories (optional). When omitted the repository is considered “local” as it is not connected to an external repository, where packages may be installed by file or managed manually.
`--access-token` *`ACCESS_TOKEN`*	

	The access token to use for remote repositories which require a token.
`--source` *`SOURCE`*	The type of source in ('USER', 'SYSTEM'). System repositories are managed outside of Blender and are considered read-only.
`--cache` *`BOOLEAN`*	Use package cache (default=1).
`--clear-all`	Clear all repositories before adding, simplifies test setup.
`--no-prefs`	Treat the user-preferences as read-only, preventing updates for operations that would otherwise modify them. This means removing extensions or repositories for example, wont update the user-preferences.

Subcommand: `repo-remove`

usage:

```
blender --command extension repo-remove [-h] [--no-prefs] ID
```

Remove a repository.

positional arguments:

ID::	The repository identifier.

options:

`-h, --help`	show this help message and exit
`--no-prefs`	Treat the user-preferences as read-only, preventing updates for operations that would otherwise modify them. This means removing extensions or repositories for example, wont update the user-preferences.

Extension Creation

Subcommand: `build`

usage:

```
blender --command extension build [-h] [--source-dir
SOURCE_DIR]
                                  [--output-dir OUTPUT_DIR]
                                  [--output-filepath
OUTPUT_FILEPATH]
                                  [--valid-tags
VALID_TAGS_JSON]
                                  [--split-platforms] [--
verbose]
```

Build a package in the current directory.

options:

`-h, --help` show this help message and exit

`--source-dir` *`SOURCE_DIR`*

The package source directory containing a `blender_manifest.toml` manifest.
Default's to the current directory.

`--output-dir` *`OUTPUT_DIR`*

The package output directory.
Default's to the current directory.

`--output-filepath` *`OUTPUT_FILEPATH`*

The package output filepath (should include a `.zip` extension).
Defaults to `{id}-{version}.zip` using values from the manifest.

`--valid-tags` *`VALID_TAGS_JSON`*

Reference a file path containing valid tags lists.
If you wish to reference custom tags a `.json` file can be used. The contents must be a

dictionary of lists where the `key` matches the extension type.

For example:

```
{"add-ons": ["Example",
"Another"], "theme": ["Other",
"Tags"]}
```

To disable validating tags, pass in an empty path `--valid-tags=""`.

`--split-platforms`	Build a separate package for each platform. Adding the platform as a file name suffix (before the extension). This can be useful to reduce the upload size of packages that bundle large platform-specific modules (`*.whl` files).
`--verbose`	Include verbose output.

Subcommand: `validate`

usage:

```
blender --command extension validate [-h]
                                     [--valid-tags
VALID_TAGS_JSON]
                                     [SOURCE_PATH]
```

Validate the package meta-data in the current directory.

positional arguments:

SOURCE_PATH::	The package source path (either directory containing package files or the package archive). This path must containing a `blender_manifest.toml` manifest. Defaults to the current directory.

options:

`-h, --help`
show this help message and exit

`--valid-tags` *`VALID_TAGS_JSON`*
Reference a file path containing valid tags lists.
If you wish to reference custom tags a `.json` file can be used. The contents must be a dictionary of lists where the `key` matches the extension type.

For example:

```
{"add-ons": ["Example",
"Another"], "theme": ["Other",
"Tags"]}
```

To disable validating tags, pass in an empty path `--valid-tags=""`.

Subcommand: `server-generate`

usage:

```
blender --command extension server-generate [-h] --repo-dir
REPO_DIR
                                            [--repo-config
REPO_CONFIG]
                                            [--html]
                                            [--html-template
HTML_TEMPLATE_FILE]
```

Generate a listing of all packages stored in a directory. This can be used to host packages which only requires static-file hosting.

options:

`-h, --help`
show this help message and exit

`--repo-dir` *`REPO_DIR`*
The remote repository directory.

`--repo-config` *`REPO_CONFIG`*

An optional server configuration to include information which can't be detected. Defaults to `blender_repo.toml` (in the repository directory).
This can be used to defined blocked extensions, for example

```
schema_version = "1.0.0"

[[blocklist]]
id = "my_example_package"
reason = "Explanation for why this
extension was blocked"
[[blocklist]]
id = "other_extenison"
reason = "Another reason for why this
is blocked"
```

`--html`
Create a HTML file (`index.html`) as well as the repository JSON to support browsing extensions online with static-hosting.

`--html-template` *`HTML_TEMPLATE_FILE`*
An optional HTML file path to override the default HTML template with your own.
The following keys will be replaced with generated contents:

- `${body}` is replaced the extensions contents.
- `${date}` is replaced the creation date.

Rendering From The Command Line

In some situations we want to increase the render speed, access Blender remotely to render something or build scripts that use the command line.

One advantage of using the command line is that we do not need a graphical display (no need for X server on Linux for example) and consequently we can render via a remote shell (typically SSH).

- See Command Line Arguments for a full list of arguments (for example to specify which scene to render, the end frame number, etc.), or simply run:

```
blender --help
```

- See Command Line Launching for specific instructions on launching Blender from the command line.

Note

Arguments are executed in the order they are given!

The following command will not work, since the output and extension are set after Blender is told to render:

```
blender -b file.blend -a -x 1 -o //render
```

The following command will behave as expected:

```
blender -b file.blend -x 1 -o //render -a
```

Always position `-f` or `-a` as the last arguments.

Single Image

```
blender -b file.blend -f 10
```

`-b`
: Render in the background (without UI).

`file.blend`
: Path to the blend-file to render.

`-f 10`
: Render only the 10th frame.

```
blender -b file.blend -o /project/renders/frame_##### -F
OPEN_EXR -f -2
```

`-o /project/renders/frame_#####`
: Path of where to save the rendered image, using five padded zeros for the frame number.

`-F OPEN_EXR`
: Override the image format specified in the blend-file and save to an OpenEXR image.

`-f -2`
: Render only the second last frame.

Warning

Arguments are case sensitive! `-F` and `-f` are not the same.

Animation

```
blender -b file.blend -a
```

`-a`

Render the whole animation using all the settings saved in the blend-file.

```
blender -b file.blend -E CYCLES -s 10 -e 500 -t 2 -a
```

`-E CYCLES`
: Use the “Cycles Render” engine. For a list of available render engines, run `blender -E help`.

`-s 10 -e 500`
: Set the start frame to `10` and the end frame to `500`.

`-t 2`
: Use only two threads.

Cycles

In addition to the options above, which apply to all render engines, Cycles has additional options to further control its behavior. See Cycles Render Options

Scripting & Extending Blender

- Introduction
  - General Information
  - Getting Started
  - Extending Blender
- Scripting & Security
  - Scripts in Blend-Files
  - Controlling Script Execution
- Add-on Tutorial
  - Intended Audience
  - Documentation Links
  - What is an Add-on?
  - Your First Add-on
  - Your Second Add-on
  - Conclusions

Introduction

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes. Python combines remarkable power with very clear syntax.

Python scripts are a versatile way to extend Blender functionality. Most areas of Blender can be scripted, including animation, rendering, import and export, object creation and automating repetitive tasks.

To interact with Blender, scripts can make use of the tightly integrated API.

General Information

Links that are useful while writing scripts:

- Python.org – General information about Python.
- Blender Python API – Official API documentation. Use this for referencing while writing scripts.
- API Introduction – A short introduction to get you started with the API. Contains examples.

Links that deal with distributing your scripts:

- Sharing scripts – Information on how to share your scripts and get them included in the official Blender distribution.
- Creating Add-ons – Add-ons are used to encapsulate and distribute scripts.
- Add-ons project – Project to maintain a central repository of extensions to Blender.

Getting Started

Manual links

The following links take you from the basics to the more advanced concepts of Python scripting for Blender.

- Text Editor
- Python Console
- Info Editor

External links

Here are external links containing a lot of good information to start learning how to write scripts for Blender:

- Python API: Quickstart
- CG Cookie: Blender 2.8 Python Scripting Superpowers for Non-Programmers
- Olav3D Tutorials: 3D Programming for Beginners Using Python
- Blender Artists: Python Support Forum

Extending Blender

Add-ons

Add-ons are scripts that enable Blender to gain extra functionality; they can be enabled from the Preferences.

Outside of the Blender executable, there are hundreds of add-ons written by many people:

- Officially supported add-ons are bundled with Blender.
- Other add-ons are provided by Blender Extensions which aren't part of official releases. Many of them work reliably and are very useful but are not yet ensured to be stable for release.

See also

See Add-ons for documentation on add-ons included with Blender.

Scripts

Apart from add-ons, there are several other types of scripts that extend Blender’s functionality:

Modules::	Utility libraries for import into other scripts.
Presets::	Settings for Blender’s tools and key configurations.
Startup::	These files are imported when starting Blender. They define most of Blender’s UI, as well as some additional core operators.
Custom Scripts::	In contrast to add-ons, they are typically intended for one-time execution via the Text Editor.

Saving your own Scripts

File Location

All scripts are loaded from the `scripts` folder of the local, system and user paths.

You can setup an additional search path for scripts in File Paths *Preferences ‣ File Paths*.

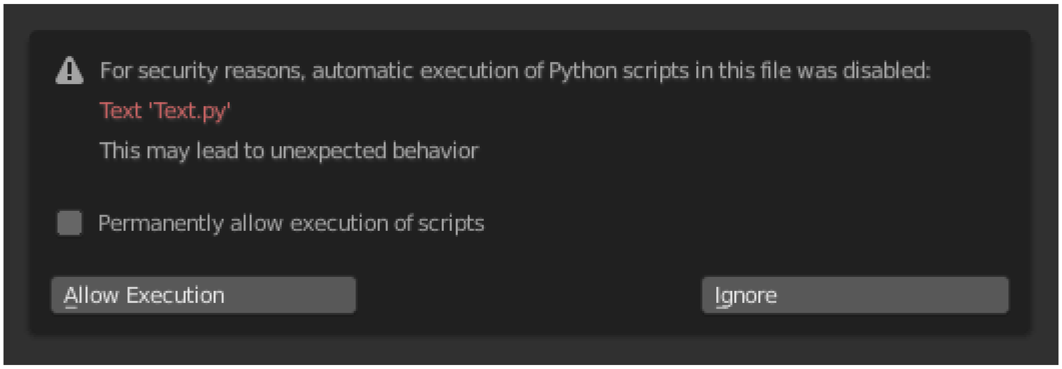
Installation

Add-ons are conveniently installed through Blender in the Preferences. Click the *Install…* button and select the `.zip` file.

Scripting & Security

The ability to include Python scripts within blend-files is valuable for advanced tasks such as rigging and automation. However, it poses a security risk since Python does not restrict what a script can do. Therefore, you should only run scripts from sources you know and trust. Automatic execution is disabled by default; however, some blend-files need this to function properly.

When a blend-file tries to execute a script and is not allowed, a dialog will appear. In it, you can choose to *Allow Execution* or to *Ignore* the scripts.



For security reasons, automatic execution of Python scripts in this file was disabled:

Text 'Text.py'

This may lead to unexpected behavior

☐ Permanently allow execution of scripts

An Auto-run warning in the Info editor's header.

Scripts in Blend-Files

Auto Execution

Here are the different ways blend-files may automatically run scripts.

Registered Text-Blocks

A text data-block can have its *Register* option enabled which means it will load on start.

Animation Drivers

Python expressions can be used to *Drive* values and are often used in more advanced rigs and animations.

Manual Execution

There are other ways scripts in a blend-file may execute that require user interaction (therefore will run even when auto execution is off), but you should be aware that this is the case since it is not necessarily obvious.

- Running a script in the Text editor.
- Rendering with Freestyle, because Freestyle uses scripts to control line styles.

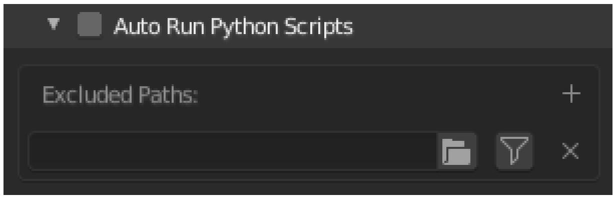
Controlling Script Execution

Blender provides a number of ways to control whether scripts from a blend-file are allowed to automatically execute.

First, the File Browser has the option **Trusted Source** which you can use on a case-by-case basis to control auto execution. Since you may forget to set this, or may open a file without going through the File Browser, you can change the default (described next).

Setting Defaults

In the Preferences, there is the toggle to Auto Run Python Scripts. This means the **Trusted Source** option in the File Browser will be enabled by default, and scripts can run when blend-files are loaded without using the File Browser. Once enabled you have the option to exclude certain directories; a typical configuration would be to trust all paths except for the download directory.



The Auto Run Python Scripts checkbox.

Command Line

You may want to perform batch rendering or some other task from the command line, running Blender without an interface. In this case, the Preferences are still used but you may want to override them:

- Enable with `-y` or `--enable-autoexec`
- Disable with `-Y` or `--disable-autoexec`

Example

To render an animation in background mode, allowing drivers and other scripts to run:

```
blender --background --enable-autoexec my_movie.blend --render-anim
```

Note

These command-line arguments can be used to start a regular Blender instance and will still override the Preferences.

Add-on Tutorial

Intended Audience

This tutorial is designed to help technical artists or developers learn to extend Blender. An understanding of the basics of Python is expected for those working through this tutorial.

Prerequisites

Before going through the tutorial you should…

- Be familiar with the basics of working in Blender.
- Know how to run a script in Blender’s Text editor.
- Have an understanding of Python primitive types (integer, Boolean, string, list, tuple, dictionary, and set).
- Be familiar with the concept of Python modules.
- Have a basic understanding of classes (object orientation) in Python.

Suggested reading before starting this tutorial.

- Dive Into Python sections (1, 2, 3, 4, and 7).
- Blender API Quickstart to help become familiar with Blender/Python basics.

To best troubleshoot any error message Python prints while writing scripts, you run Blender from a terminal. See Use The Terminal.

Tip

You can enable Developer Extras in the preferences to enable features that make developing add-ons easier.

Documentation Links

While going through the tutorial, you may want to look into our reference documentation.

- Blender API Overview: This document is rather detailed but helpful if you want to know more on a topic.
- `bpy.context` API reference – Handy to have a list of available items your script may operate on.
- `bpy.types.Operator` – The following add-ons define operators, these docs give details and more examples of operators.

What is an Add-on?

An add-on is simply a Python module with some additional requirements so Blender can display it in a list with useful information.

To give an example, here is the simplest possible add-on:

```
bl_info = {
    "name": "My Test Add-on",
    "blender": (2, 80, 0),
    "category": "Object",
}
def register():
    print("Hello World")
def unregister():
    print("Goodbye World")
```

`bl_info`

is a dictionary containing add-on metadata such as the title, version and author to be displayed in the Preferences add-on list. It also specifies the minimum Blender version required to run the script; older versions won't display the add-on in the list.

`register`

is a function which only runs when enabling the add-on, this means the module can be loaded without activating the add-on.

`unregister`

is a function to unload anything setup by `register`, this is called when the add-on is disabled.

Notice this add-on does not do anything related to Blender (the **`blender_api:bpy`** module is not imported for example).

This is a contrived example of an add-on that serves to illustrate the point that the base requirements of an add-on are simple.

An add-on will typically register operators, panels, menu items, etc, but it's worth noting that *any* script can do this, when executed from the Text editor or even the interactive console – there is nothing inherently different about an add-on that allows it to integrate with Blender, such functionality is just provided by the **`blender_api:bpy`** module for any script to access.

So an add-on is just a way to encapsulate a Python module in a way a user can easily utilize.

Note

Running this script within the Text editor won't print anything, to see the output it must be installed through the Preferences. Messages will be printed when enabling and disabling.

Your First Add-on

The simplest possible add-on above is useful as an example but not much else. This next add-on is simple but shows how to integrate a script into Blender using an `Operator` which is the typical way to define a tool accessed from menus, buttons and keyboard shortcuts.

For the first example we will make a script that simply moves all objects in a scene.

Write the Script

Add the following script to the Text editor in Blender:

```
import bpy

scene = bpy.context.scene
for obj in scene.objects:
    obj.location.x += 1.0
```

Click the Run Script button, all objects in the active scene are moved by 1.0 unit.

Write the Add-on (Simple)

This add-on takes the body of the script above, and adds it to an operator's `execute()` function.

```
bl_info = {
    "name": "Move X Axis",
    "blender": (2, 80, 0),
    "category": "Object",
}

import bpy


class ObjectMoveX(bpy.types.Operator):
    """My Object Moving Script"""      # Use this as a tooltip
for menu items and buttons.
    bl_idname = "object.move_x"        # Unique identifier for
buttons and menu items to reference.
    bl_label = "Move X by One"         # Display name in the
interface.
    bl_options = {'REGISTER', 'UNDO'}  # Enable undo for the
operator.

    def execute(self, context):        # execute() is called
when running the operator.

        # The original script
        scene = context.scene
        for obj in scene.objects:
            obj.location.x += 1.0

        return {'FINISHED'}            # Lets Blender know the
operator finished successfully.

def menu_func(self, context):
    self.layout.operator(ObjectMoveX.bl_idname)

def register():
    bpy.utils.register_class(ObjectMoveX)
    bpy.types.VIEW3D_MT_object.append(menu_func)  # Adds the
new operator to an existing menu.

def unregister():
    bpy.utils.unregister_class(ObjectMoveX)


# This allows you to run the script directly from Blender's
Text editor
# to test the add-on without having to install it.
if __name__ == "__main__":
    register()
```

Note

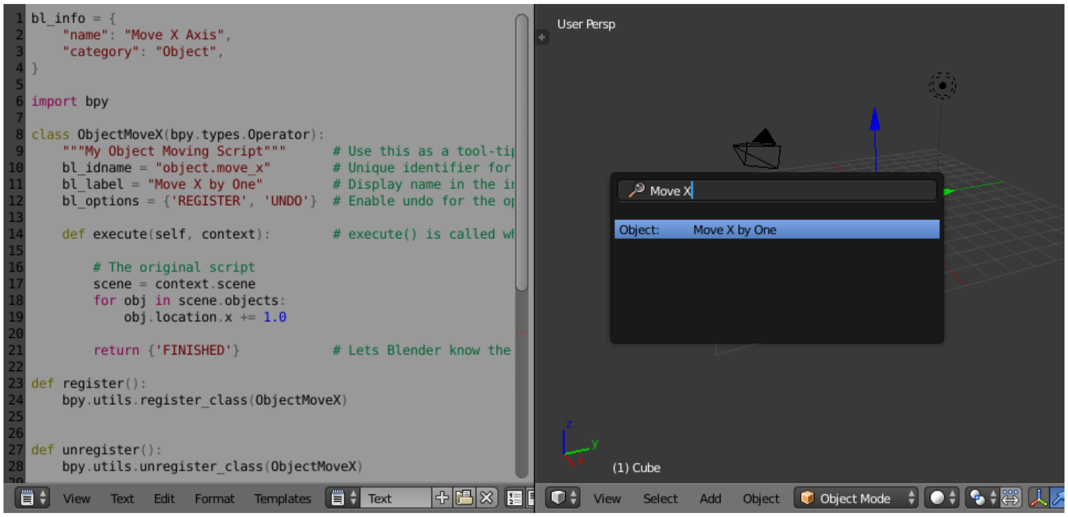
`bl_info` is split across multiple lines, this is just a style convention used to more easily add items.

Note

Rather than using `bpy.context.scene`, we use the `context.scene` argument passed to `execute()`. In most cases these will be the same. However in some cases, operators will be passed a custom context so script authors should prefer the `context` argument passed to operators.

To test the script, you can copy and paste it into Blender's Text editor and run it. This will execute the script directly and call register immediately.

However running the script won't move any objects. For this, you need to execute the newly registered operator.



```
bl_info = {
    "name": "Move X Axis",
    "category": "Object",
}

import bpy

class ObjectMoveX(bpy.types.Operator):
    """My Object Moving Script"""      # Use this as a tool-ti
    bl_idname = "object.move_x"        # Unique identifier for
    bl_label = "Move X by One"         # Display name in the i
    bl_options = {'REGISTER', 'UNDO'}  # Enable undo for the o

    def execute(self, context):        # execute() is called w

        # The original script
        scene = context.scene
        for obj in scene.objects:
            obj.location.x += 1.0

        return {'FINISHED'}            # Lets Blender know the

def register():
    bpy.utils.register_class(ObjectMoveX)


def unregister():
    bpy.utils.unregister_class(ObjectMoveX)
```

Operator Search menu.

Open the [Operator Search](#) menu and type in "Move X by One" (the `bl_label`), then `Return`.

The objects should move as before.

Keep this add-on open in Blender for the next step - Installing.

Install the Add-on

Once you have your add-on within in Blender's Text editor, you will want to be able to install it so it can be enabled in the Preferences to load on startup.

Even though the add-on above is a test, let's go through the steps anyway so you know how to do it for later.

To install the Blender text as an add-on, you will first have to save it on drive. Take care to obey the naming restrictions that apply to Python modules and end with a `.py` extension.

Once the file is on drive, you can install it as you would for an add-on downloaded online.

Open the *Preferences* ‣ *Add-ons* ‣ *Install…* and select the file.

Now the add-on will be listed and you can enable it by pressing the checkbox, if you want it to be enabled on restart, press *Save as Default*. The operator can be run in the same way as described in the previous section.

When the add-on is enabled, Blender executes the code and runs the `register()` function. When the add-on is disabled, Blender runs the `unregister()` function.

Note

The destination of the add-on depends on your Blender configuration. When installing an add-on the source and destination paths are printed in the console. You can also find add-on path locations by running this in the Python Console:

```
import addon_utils
print(addon_utils.paths())
```

More is written on this topic here: Directory Layout.

Your Second Add-on

For our second add-on, we will focus on object instancing – this is – to make linked copies of an object in a similar way to what you may have seen with the Array modifier.

Write the Script

As before, first we will start with a script, develop it, then convert it into an add-on.

```
import bpy
from bpy import context

# Get the current scene
scene = context.scene

# Get the 3D cursor location
cursor = scene.cursor.location

# Get the active object (assume we have one)
obj = context.active_object

# Now make a copy of the object
obj_new = obj.copy()

# The new object has to be added to a collection in the scene
scene.collection.objects.link(obj_new)

# Now we can place the object
obj_new.location = cursor
```

Now try copying this script into Blender and run it on the default Cube. Make sure you click to move the 3D cursor before running as the duplicate will appear at the cursor's location.

After running, notice that when you go into *Edit Mode* to change the Cube – all of the copies change. In Blender, this is known as *Linked Duplicates*.

Next, we're going to do this in a loop, to make an array of objects between the active object and the cursor.

```
import bpy
from bpy import context

scene = context.scene
cursor = scene.cursor.location
obj = context.active_object

# Use a fixed value for now, eventually make this user
adjustable
total = 10
```

```
# Add 'total' objects into the scene
for i in range(total):
    obj_new = obj.copy()
    scene.collection.objects.link(obj_new)

    # Now place the object in between the cursor
    # and the active object based on 'i'
    factor = i / total
    obj_new.location = (obj.location * factor) + (cursor * (1.0
- factor))
```

Try running this script with the active object and the cursor spaced apart to see the result.

With this script you'll notice we're doing some math with the object location and cursor, this works because both are 3D `mathutils.Vector` instances, a convenient class provided by the `mathutils` module which allows vectors to be multiplied by numbers and matrices.

If you are interested in this area, read into `mathutils.Vector` – there are many handy utility functions such as getting the angle between vectors, cross product, dot products as well as more advanced functions in `mathutils.geometry` such as Bézier spline interpolation and ray-triangle intersection.

For now we will focus on making this script an add-on, but it's good to know that this 3D math module is available and can help you with more advanced functionality later on.

Write the Add-on

The first step is to convert the script as-is into an add-on:

```
bl_info = {
    "name": "Cursor Array",
    "blender": (2, 80, 0),
    "category": "Object",
}

import bpy
```

```
class ObjectCursorArray(bpy.types.Operator):
    """Object Cursor Array"""
    bl_idname = "object.cursor_array"
    bl_label = "Cursor Array"
    bl_options = {'REGISTER', 'UNDO'}

    def execute(self, context):
        scene = context.scene
        cursor = scene.cursor.location
        obj = context.active_object

        total = 10

        for i in range(total):
            obj_new = obj.copy()
            scene.collection.objects.link(obj_new)

            factor = i / total
            obj_new.location = (obj.location * factor) +
(cursor * (1.0 - factor))

        return {'FINISHED'}

def register():
    bpy.utils.register_class(ObjectCursorArray)


def unregister():
    bpy.utils.unregister_class(ObjectCursorArray)


if __name__ == "__main__":
    register()
```

Everything here has been covered in the previous steps, you may want to try run the add-on still and consider what could be done to make it more useful.

The two of the most obvious missing things are – having the total fixed at 10, and having to access the operator with Operator Search is not very convenient.

Both these additions are explained next, with the final script afterwards.

Operator Property

There are a variety of property types that are used for tool settings, common property types include: int, float, vector, color, Boolean and string.

These properties are handled differently to typical Python class attributes because Blender needs to display them in the interface, store their settings in keymaps and keep settings for reuse.

While this is handled in a fairly Pythonic way, be mindful that you are in fact defining tool settings that are loaded into Blender and accessed by other parts of Blender, outside of Python.

To get rid of the literal 10 for `total`, we'll use an operator property. Operator properties are defined via bpy.props module, this is added to the class body:

```
# moved assignment from execute() to the body of the class...
total: bpy.props.IntProperty(name="Steps", default=2, min=1,
max=100)

# and this is accessed on the class
# instance within the execute() function as...
self.total
```

These properties from **`bpy.props`** are handled specially by Blender when the class is registered so they display as buttons in the user interface. There are many arguments you can pass to properties to set limits, change the default and display a tooltip.

See also

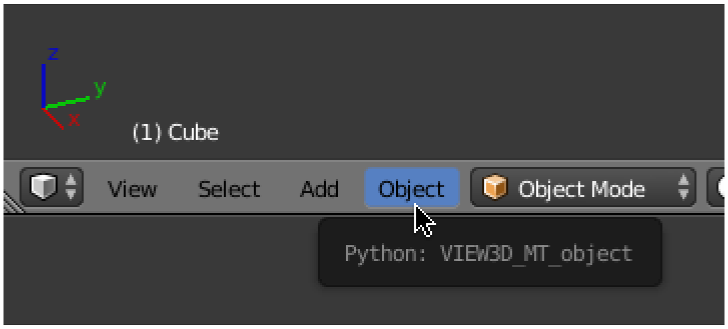
`bpy.props.IntProperty()`

This document doesn't go into details about using other property types. However, the link above includes examples of more advanced property usage.

Menu Item

Add-ons can add to the user interface of existing panels, headers and menus defined in Python.

For this example we'll add to an existing menu.



Python: VIEW3D_MT_object

Menu Identifier.

To find the identifier of a menu, first enable *Python Tooltips* in the preferences. Then you can hover your mouse over the menu item and the identifier is displayed.

The method used for adding a menu item is to append a draw function into an existing class:

```
def menu_func(self, context):
    self.layout.operator(ObjectCursorArray.bl_idname)

def register():
    bpy.utils.register_class(ObjectCursorArray)
    bpy.types.VIEW3D_MT_object.append(menu_func)
```

For docs on extending menus, see: `bpy.types.Menu`.

Keymap

In Blender, add-ons have their own keymaps so as not to interfere with Blender’s built-in keymaps.

In the example below, a new object mode `bpy.types.KeyMap` is added, then a `bpy.types.KeyMapItem` is added to the keymap which references our newly added operator, using `Shift-Ctrl-T` as the key shortcut to activate it.

```
# store keymaps here to access after registration
addon_keymaps = []

def register():

    # handle the keymap
    wm = bpy.context.window_manager
    km = wm.keyconfigs.addon.keymaps.new(name='Object Mode',
space_type='EMPTY')

    kmi = km.keymap_items.new(ObjectCursorArray.bl_idname, 'T',
'PRESS', ctrl=True, shift=True)
    kmi.properties.total = 4

    addon_keymaps.append((km, kmi))


def unregister():

    # handle the keymap
    for km, kmi in addon_keymaps:
        km.keymap_items.remove(kmi)
    addon_keymaps.clear()
```

Notice how the keymap item can have a `total` setting different than the default set by the operator, this allows you to have multiple keys accessing the same operator with different settings.

Note

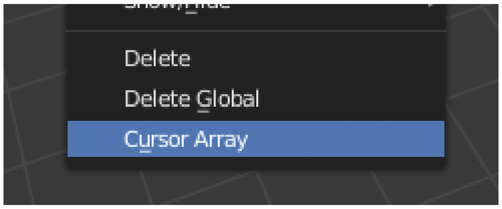
While `Shift-Ctrl-T` is not a default Blender key shortcut, it is hard to make sure add-ons will not overwrite each other’s keymaps. Thus at least take care when assigning keys that they do not conflict with important functionality of Blender (see also is key free add-on).

For API documentation on the functions listed above, see:

- `bpy.types.KeyMaps.new()`
- `bpy.types.KeyMap`
- `bpy.types.KeyMapItems.new()`
- `bpy.types.KeyMapItem`

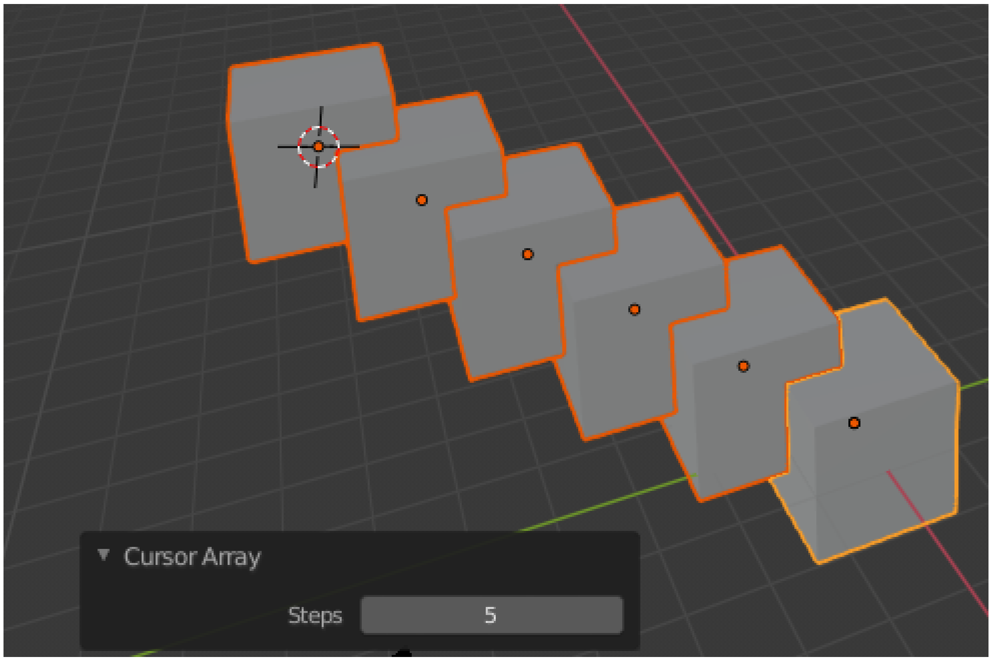
Bringing It All Together

```
bl_info = {
    "name": "Cursor Array",
    "blender": (2, 80, 0),
    "category": "Object",
}

import bpy


class ObjectCursorArray(bpy.types.Operator):
    """Object Cursor Array"""
    bl_idname = "object.cursor_array"
    bl_label = "Cursor Array"
    bl_options = {'REGISTER', 'UNDO'}

    total: bpy.props.IntProperty(name="Steps", default=2,
min=1, max=100)

    def execute(self, context):
        scene = context.scene
        cursor = scene.cursor.location
        obj = context.active_object

        for i in range(self.total):
            obj_new = obj.copy()
            scene.collection.objects.link(obj_new)
            factor = i / self.total
            obj_new.location = (obj.location * factor) +
(cursor * (1.0 - factor))

        return {'FINISHED'}


def menu_func(self, context):
    self.layout.operator(ObjectCursorArray.bl_idname)

# store keymaps here to access after registration
addon_keymaps = []


def register():
    bpy.utils.register_class(ObjectCursorArray)
    bpy.types.VIEW3D_MT_object.append(menu_func)

    # handle the keymap
    wm = bpy.context.window_manager
    # Note that in background mode (no GUI available),
keyconfigs are not available either,
    # so we have to check this to avoid nasty errors in
background case.
    kc = wm.keyconfigs.addon
    if kc:
        km = wm.keyconfigs.addon.keymaps.new(name='Object
Mode', space_type='EMPTY')
        kmi = km.keymap_items.new(ObjectCursorArray.bl_idname,
'T', 'PRESS', ctrl=True, shift=True)
        kmi.properties.total = 4
        addon_keymaps.append((km, kmi))

def unregister():
    # Note: when unregistering, it's usually good practice to
do it in reverse order you registered.
    # Can avoid strange issues like keymap still referring to
operators already unregistered...
    # handle the keymap
    for km, kmi in addon_keymaps:
        km.keymap_items.remove(kmi)
    addon_keymaps.clear()
    bpy.utils.unregister_class(ObjectCursorArray)
    bpy.types.VIEW3D_MT_object.remove(menu_func)


if __name__ == "__main__":
    register()
```



In the menu.

Run the script (or save it and add it through the Preferences like before) and it will appear in the *Object* menu.



Cursor Array

Steps 5

Operator Property.

After selecting it from the menu, you can choose how many instances of the cube you want create.

Note

Directly executing the script multiple times will add the menu each time too. While not useful behavior, there is nothing to worry about since add-ons will not register themselves multiple times when enabled through the Preferences.

Conclusions

Add-ons can encapsulate certain functionality neatly for writing tools to improve your workflow or for writing utilities for others to use.

While there are limits to what Python can do within Blender, there is certainly a lot that can be achieved without having to dive into Blender's C/C++ code.

The example given in the tutorial is limited, but shows the Blender API used for common tasks that you can expand on to write your own tools.

Further Reading

Blender comes with commented templates which are accessible from the Text editor's header. If you have specific areas you want to see example code for, this is a good place to start.

Here are some sites you might like to check on after completing this tutorial.

- Blender/Python API Overview – For more background details on Blender/Python integration.
- How to Think Like a Computer Scientist – Great info for those who are still learning Python.
- Blender Development – Blender Development, general information and helpful links.
- Blender Developer Forum – Forum where people ask Python development questions.

Creating Extensions

Extensions are **add-ons** or **themes** used to extend the core functionality of Blender. They are shared in online platforms, and can be installed and updated from within Blender.

The official extensions platform for the Blender project is extensions.blender.org. Other third party sites can also be supported, as long as they follow the Extensions Platform specification.

- Getting started
- Compatible licenses
- Supported tags
- Add-ons
- Python Wheels
- Creating a Repository

See also

For extension add-on guidelines (requirements for `extensions.blender.org`), refer to the Developer Handbook.

For the extension settings, and how to manage them, refer to the User Preferences.

For managing extensions from the command line, refer to Extension Command Line Arguments.

How to Create Extensions

Creating an extension takes only a few steps:

1. Open the directory containing the add-on code or theme file.
2. Add a blender_manifest.toml file with all the required meta-data `(name, maintainer, ...)`.
3. Use the Blender command-line tool to build the extension .zip file.

How to publish to the Blender Extensions Platform:

- Install from Disk to test if everything is working well.
- Upload the .zip file (this step requires Blender ID).

The extension will be held for review, and published once the moderation team approves it.

Extension files

An extension is shared as a `.zip` archive containing a manifest file and other files. The expected files depend on the extension type.

Add-on extension

Add-ons need at least the manifest and an `__init__.py` file, while more complex add-ons have a few different .py files or wheels together.

```
my_extension-0.0.1.zip
├─ __init__.py
├─ blender_manifest.toml
└─ (...)
```

Theme extension

A theme extension only needs the manifest and the .xml theme file.

```
my_extension-0.0.1.zip
├─ blender_manifest.toml
└─ theme.xml
```

Note

Extensions can optionally have all its files inside a folder (inside the archive). This is a common behavior when saving a repository as ZIP from version-control platforms.

Manifest

A manifest is a file with all the meta-data required for an extension to be processed. This example is a good starting point to the `blender_manifest.toml` that should be inside the `.zip`.

```
schema_version = "1.0.0"

# Example of manifest file for a Blender extension
# Change the values according to your extension
id = "my_example_extension"
version = "1.0.0"
name = "My Example Extension"
tagline = "This is another extension"
maintainer = "Developer name <email@address.com>"
# Supported types: "add-on", "theme"
type = "add-on"

# # Optional: link to documentation, support, source files, etc
# website = "https://extensions.blender.org/add-ons/my-example-package/"

# # Optional: tag list defined by Blender and server, see:
# # https://docs.blender.org/manual/en/dev/advanced/extensions/tags.html
# tags = ["Animation", "Sequencer"]

blender_version_min = "4.2.0"
```

```
# # Optional: Blender version that the extension does not
support, earlier versions are supported.
# # This can be omitted and defined later on the extensions
platform if an issue is found.
# blender_version_max = "5.1.0"

# License conforming to https://spdx.org/licenses/ (use "SPDX:
prefix)
#
https://docs.blender.org/manual/en/dev/advanced/extensions/lice
nses.html
license = [
  "SPDX:GPL-3.0-or-later",
]
# # Optional: required by some licenses.
# copyright = [
#   "2002-2024 Developer Name",
#   "1998 Company Name",
# ]

# # Optional: list of supported platforms. If omitted, the
extension will be available in all operating systems.
# platforms = ["windows-x64", "macos-arm64", "linux-x64"]
# # Other supported platforms: "windows-arm64", "macos-x64"

# # Optional: bundle 3rd party Python modules.
# #
https://docs.blender.org/manual/en/dev/advanced/extensions/pyth
on_wheels.html
# wheels = [
#   "./wheels/hexdump-3.3-py3-none-any.whl",
#   "./wheels/jsmin-3.0.1-py3-none-any.whl",
# ]

# # Optional: add-ons can list which resources they will
require:
# # * files (for access of any filesystem operations)
# # * network (for internet access)
# # * clipboard (to read and/or write the system clipboard)
# # * camera (to capture photos and videos)
# # * microphone (to capture audio)
# #
# # If using network, remember to also check
```

```
`bpy.app.online_access`
# #
https://docs.blender.org/manual/en/dev/advanced/extensions/addo
ns.html#internet-access
# #
# # For each permission it is important to also specify the
reason why it is required.
# # Keep this a single short sentence without a period (.) at
the end.
# # For longer explanations use the documentation or detail
page.
#
# [permissions]
# network = "Need to sync motion-capture data to server"
# files = "Import/export FBX from/to disk"
# clipboard = "Copy and paste bone transforms"

# # Optional: advanced build settings.
# #
https://docs.blender.org/manual/en/dev/advanced/extensions/comm
and_line_arguments.html#command-line-args-extension-build
# [build]
# # These are the default build excluded patterns.
# # You only need to edit them if you want different options.
# paths_exclude_pattern = [
#   "__pycache__/",
#   "/.git/",
#   "/*.zip",
# ]
```

Required values:

blender_version_min::	Minimum supported Blender version - use at least `4.2.0`.
id::	Unique identifier for the extension.
license::	List of licenses, use SPDX license identifier.
maintainer::	Maintainer of the extension.
name::	Complete name of the extension.
schema_version::	Internal version of the file format - use `1.0.0`.
tagline::	

	One-line short description, up to 64 characters - cannot end with punctuation.
type::	“add-on”, “theme”.
version::	Version of the extension - must follow semantic versioning.

Optional values:

blender_version_max::	Blender version that the extension does not support, earlier versions are supported.
website::	Website for the extension.
copyright::	Some licenses require a copyright, copyrights must be “Year Name” or “Year-Year Name”.
tags::	List of tags. See the list of available tags.
platforms::	List of supported platforms. If omitted, the extension will be available in all operating systems. The available options are [“windows-x64”, “windows-arm64”, “macos-x64”, “macos-arm64”, “linux-x64”]
wheels::	List of relative file-paths Python Wheels.
permissions::	Add-ons can list which resources they require. The available options are *files*, *network*, *clipboard*, *camera*, *microphone*. Each permission should be followed by an explanation (short single-sentence, up to 64 characters, with no end punctuation).

Optional values for “build”:

These values are only used by the build sub-command.

paths::	A list of file-paths relative to the manifest to include when building the package.
paths_exclude_pattern::	A list of file-path patterns to exclude include when building the package.

The pattern matching is compatible with gitignore.
Note that setting this value isn't supported when `paths` is also declared.

If the `[build]` table isn't declared the following default is used:

```
[build]
paths_exclude_pattern = [
  "__pycache__/",
  ".*",
  "*.zip",
]
```

Reserved:

These values **must not** be declared in a TOML and are reserved for internal use.

- `[build.generated]`

Note

All the values present in the manifest file must be filled (i.e., cannot be empty, nor text `""`, nor list `[]`).

If you don't want to set one of the optional values just exclude it from the manifest altogether.

Command-line

Extensions can be built, validated & installed via command-line.

To build the package defined in the current directory use the following commands:

```
blender --command extension build
```

See build docs.

—

To validate the manifest without building the package:

```
blender --command extension validate
```

You may also validate a package without having to extract it first.

```
blender --command extension validate add-on-package.zip
```

See validate docs.

See also

Extensions Command Line Arguments.

Third party extension sites

If you want to host the extensions yourself, see the Creating an Extensions Repository docs.

Extension Licenses

The Blender Extensions Platform only supports free and open source extensions compliant with the Blender's license:

- For add-ons, the required license is GNU General Public License v3.0 or later.
- For themes, the recommended license is GNU General Public License v3.0 or later, but any GPL-compatible license is supported.
- For assets used in add-ons, the required license is Public Domain (CC0).

Extensions Tags

A different set of tags is available for the different extensions types. This is the list of the tags currently supported by the Blender Extensions Platform:

Add-ons

- 3D View
- Add Curve
- Add Mesh
- Animation
- Bake
- Camera
- Compositing
- Development
- Game Engine
- Geometry Nodes
- Grease Pencil
- Import-Export
- Lighting
- Material
- Modeling
- Mesh
- Node
- Object
- Paint
- Pipeline
- Physics
- Render
- Rigging
- Scene
- Sculpt
- Sequencer
- System

- Text Editor
- Tracking
- User Interface
- UV

Themes

- Dark
- Light
- Colorful
- Inspired By
- Print
- Accessibility
- High Contrast

Add-ons

Add-ons let you extend Blender's functionality using Python. Most of the time you can get add-ons as part of the Extensions system.

Tip

If the Add-on does not activate when enabled, check the Console window for any errors that may have occurred.

Internet Access

If the add-on needs to use internet, it must check for the (read-only) property `bpy.app.online_access`. This option is controlled by Preferences, which can be overriding via command-line (`--offline-mode` / `--online-mode`).

For better error messages, you can check also for `bpy.app.online_access_overriden`, to determine whether users can turn Allow Online Access on the preferences, or not.

Note

Add-ons that follow this setting will only connect to the internet if enabled. However, Blender cannot prevent third-party add-ons from violating this rule.

Bundle Dependencies

For add-ons to be bundled as extensions, they must be self-contained. That means they must come with all its dependencies. In particular 3rd party

Python modules.

There are a few options for this:

Bundle with Python Wheels.

This is the recommended way to bundle dependencies.

Bundle other add-ons together.

This is recommended if an add-on depends on another add-on.

Make sure that both the individual and the combined add-on check for already registered types (Operators, Panels, …). This avoids duplication of operators and panels on the interface if the add-ons are installed as a bundle and individually.

Bundle with Vendorize

This can be used as a way to bundle a pure Python dependencies as a sub-module.

This has the advantage of avoiding version conflicts although it requires some work to setup each package.

Local Storage

Add-ons must not assume their own directory is user writable since this may not be the case for “System” repositories. Writing files into the add-on’s directory also has the down side that upgrading the extension will remove all files.

Add-ons which need their own user directory should use a utility function provided for this purpose:

```
extension_directory =
bpy.utils.extension_path_user(__package__, path="",
create=True)
```

If you wish create subdirectories, this can be done with the `path` argument.

This directory will be kept between upgrades but will be removed if the extension is uninstalled.

Legacy vs Extension Add-ons

With the introduction of Extensions in Blender 4.2, the old way of creating add-ons is considered deprecated. While the changes are rather small they impact existing add-ons.

In order to allow a smooth transition process, the so-called legacy add-ons will continue to be supported by Blender. They can be installed via Install legacy Add-on button in the User Preferences.

All add-on maintainers are urged to convert the add-ons they want to share, so they are future proof and can support features like updating from the extensions platform.

Converting a Legacy Add-on into an Extension

1. Create a manifest file.
2. Remove the `bl_info` information (this is now in the manifest).
3. Replace all references to the module name to `__package__`.
4. Make all module imports to use relative import.
5. Use wheels to pack your external Python dependencies.
6. Remember to test it thoroughly.

Note

For testing it is important to install the extension from disk and check if everything is working well. This will get you as close to the final experience as possible.

Extensions and Namespace

The legacy add-ons would use their module name to access the preferences. This could lead to a name clash when extensions with the same name (from different repositories) would be installed. To prevent this conflict, the repository name is now part of the namespace.

For example, now instead of `kitsu` the module name would be `bl_ext.{repository_module_name}.kitsu` instead.

This has a few implications for preferences and module imports.

User Preferences and `__package__`

Add-ons can define their own preferences which use the package name as an identifier. This can be accessed using `__package__`.

This was already supported in the legacy add-ons, but not reinforced. As such this can break backward compatibility.

Before:

```
class KitsuPreferences(bpy.types.AddonPreferences):
    bl_idname = "kitsu"
    # ... snip ...

# Access with:
addon_prefs = bpy.context.preferences.addons["kitsu"]
```

Now:

```
class KitsuPreferences(bpy.types.AddonPreferences):
    bl_idname = __package__
    # ... snip ...

# Access with:
addon_prefs = bpy.context.preferences.addons[__package__]
```

Sub-packages

An add-on that defines sub-packages (sub-directories with their own `__init__.py` file) that needs to use this identifier will have to import the top-level package using a relative import.

```
from .. import __package__ as base_package
```

Then `base_package` can be used instead of `__package__`. The `..` imports relative to the packages parent, sub-sub-packages must use `...` and so on.

Note

- The value of `__package__` will vary between systems so it should never be replaced with a literal string.
- Extensions must not manipulate the value of `__package__` as this may cause unexpected behavior or errors.

Relative Imports

before::	`from kitsu import utils`
now::	`from . import utils`

Importing packages within the add-on module need to use relative paths. This is a standard Python feature and only applicable for add-ons that have multiple folders.

This was already supported in the legacy add-ons, but not reinforced. As such this can break backward compatibility.

Python Wheels

Python wheels (`*.whl`) are the standard way of distributing Python modules. They are supported in Blender to make self-contained Python Extensions.

Guidelines

- By convention, always locate the files under `./wheels/`.

Requirements

- Wheels must be bundled unmodified from Python's package index.
- Wheels must include their dependencies.
- Wheels filenames must match Python's binary distribution specification: see docs. *Wheels downloaded from Python's package index will follow this convention.*
- Use forward slashes as path separators when listing them on the manifest.

How to Bundle Wheels

Python wheels (`*.whl`) can be bundled using the following steps.

Downloading Wheels

Download the wheel to the directory `./wheels/`.

For wheels that are platform independent this example downloads `jsmin`:

```
pip wheel jsmin -w ./wheels
```

For wheels that contain binary compiled files, wheels for all supported platforms should be included:

This example downloads `pillow` - the popular image manipulation module.

```
pip download pillow --dest ./wheels --only-binary=:all: --
python-version=3.11 --platform=macosx_11_0_arm64
pip download pillow --dest ./wheels --only-binary=:all: --
python-version=3.11 --platform=manylinux_2_28_x86_64
pip download pillow --dest ./wheels --only-binary=:all: --
python-version=3.11 --platform=win_amd64
```

The available platform identifiers are listed on pillow's download page.

Update the Manifest

In `blender_manifest.toml` include the wheels as a list of paths, e.g.

```
wheels = [
   "./wheels/pillow-10.3.0-cp311-cp311-
macosx_11_0_arm64.whl",
   "./wheels/pillow-10.3.0-cp311-cp311-
manylinux_2_28_x86_64.whl",
   "./wheels/pillow-10.3.0-cp311-cp311-win_amd64.whl",
]
```

Now installing the package will extract the wheel into the extensions own `site-packages` directory.

Running

Once the extension has been installed you can check the module is being loaded by importing it in the Python console and printing it's location:

```
import PIL
print(PIL.__file__)
```

Platform Builds

Wheels can severely impact the size of an extension. To mitigate this, it is possible to build different extension zip files for each unique required platform.

For this you need to use the `--split-platforms` option from the build command.

```
blender --command extension build --split-platforms
```

Example

Manifest file excerpt:

```
id = "my_addon_with_wheels"
version = "1.0.0"

platforms = ["windows-x64", "macos-x64"]
wheels = [
   "./wheels/pillow-10.3.0-cp311-cp311-macosx_11_0_arm64.whl",
   "./wheels/pillow-10.3.0-cp311-cp311-
manylinux_2_28_x86_64.whl",
   "./wheels/pillow-10.3.0-cp311-cp311-win_amd64.whl",
]
```

Generated files from `--split-platforms`:

- my_addon_with_wheels-1.0.0-windows_x64.zip
- my_addon_with_wheels-1.0.0-macos_x64.zip

Note

Even though there is a Linux-only wheel present, no Linux zip file is generated. This happens because the `platforms` field only has Mac and Windows.

Creating an Extensions Repository

Third party sites that wish to support extensions in Blender can do so in several ways, in order of complexity:

- Static: Host a static JSON file listing all the packages of your repository.
- Dynamic: Serve the JSON file on-demand based on the Blender version and platform.
- Platform: Fork the entire Extensions Website to create your own.

Tags and Translations

If you are planning to use a different set of tags than the ones used by Blender Extensions Platform, remember to submit a pull request to tags.py.

This way they can be shown translated within Blender.

Sub-Pages

- Static Repository
- Dynamic Repository

Creating a Static Extensions Repository

To host your own extensions and leverage Blender update system all that is required is to host a static JSON file that is generated by Blender.

JSON

To generate a valid JSON file use the server generate Blender command-line tool:

```
blender --command extension server-generate --repo-
dir=/path/to/packages
```

This creates an `index.json` listing from all the .zip extensions found in the *–repo-dir* location.

For more details, read the generated JSON API.

Testing

To test the generated repository, add a new Remote Repository from the Preferences:

- Get Extensions → Repositories → [+] → Add Remote Repository
- Paste the location of the generated JSON as the URL. So the example `/path/to/packages` would use the:
  - `file:///path/to/packages/index.json` on Linux and macOS.
  - `file:///C:/path/to/packages/index.json` on Windows.
  - `file://HOST/share/path/to/packages/index.json` for network shares on Windows.

Tip

Open `file:///` in a web browser and navigate to the repository location and copy that as the remote repository URL.

Extensions Listing HTML

The `server-generate` command can optionally create a simple website using the `--html` argument.

```
blender --command extension server-generate --repo-
dir=/path/to/packages --html
```

This creates an `index.html` file ready to use, listing extensions which can be dropped into Blender for installation.

Download Links

In order to support drag and drop for installing from a remote repository, there are a few optional ways to prepare the URLs.

The only strict requirement is that the download URL must end in `.zip`.

You can pass different arguments to the URL to give more clues to Blender about what to do with the dropped URL.

repository::	Link to the JSON file to be used to install the repository in Blender. Supports relative URLs.
platforms::	Comma-separated list of supported platforms. If omitted, the extension will be available in all operating systems.
blender_version_min::	Minimum supported Blender version, e.g., `4.2.0`.
blender_version_max::	Blender version that the extension does not support, earlier versions are supported.

Tip

The more details you provide, the better the user experience.

With the exception of the `repository`, all the other parameters can be extracted from the extensions manifest. Those arguments are to be encoded as part of the URL.

Expected format:

```
{URL}.zip?repository={repository}&blender_version_min={version_min}&blender_max={version_max_exclusive}&platforms={platform1,platform2}
```

Example from self-hosted repository:

```
http://my-site.com/my-addon.zip?repository=.%2Findex.json&blender_version_min=4.2.0&platforms=windows-x64
```

Example from the Extensions Platform:

```
https://extensions.blender.org/download/sha256:57a6a5f39fa2cc834dc086a27b7b2e572c12fd14f8377fb8bd1c7022df3d7ccb/add-on-amaranth-v1.0.23.zip?repository=%2Fapi%2Fv1%2Fextensions%2F&blender_version_min=4.2.0&platforms=linux-x64%2Cmacos-x64
```

Note

`%2F` and `%2C` are simply the url-encoded equivalent of `/` and `,` respectively.

Creating a Dynamic Extensions Repository

A dynamic repository allows you to serve a smaller JSON file with only the latest version of the extensions which are compatible with the query parameters. This is only relevant for repositories which contain multiple version of multiple extensions.

For small or personal repositories it is simpler and recommended to use static repositories instead.

Listing

To setup a dynamic extensions repository, follow the steps for static repositories, since the format and the listing are the same.

Query Parameters

When Blender fetches the extensions listing it passes the following arguments to make sure only compatible extensions are listed:

- `platform`
- `blender_version`

These arguments are passed as parameters to the server via a query URL:

URL::	`https://extensions.blender.org/api/v1/extensions/`
query URL::	`https://extensions.blender.org/api/v1/extensions/?blender_version=4.2.0&platform=linux-x64`

Access Token

Some repositories may require authentication. The user can specify an `access token` for a repository, which is passed along with the API request from Blender.

This is passed to the servers via an Authorization header:

```
curl -i https://extensions.blender.org/api/v1/extensions/ \
     -H "Accept: application/json" \
     -H "Authorization: Bearer abc29832befb92983423abcaef93001"
```

Application Templates

Usage

Application templates are a feature that allows you to define a re-usable configuration that can be selected to replace the default configuration, without requiring a separate Blender installation or overwriting your personal settings.

Application templates can be selected from the splash screen or *File* ‣ *New* submenu. When there are no templates found the menu will not be displayed on the splash screen.

New application templates can be installed from the Blender Menu. If you would like to keep the current application template active on restarting Blender, save your preferences.

Motivation

In some cases it's not enough to write a single script or add-on, and expect someone to replace their preferences and startup file, install scripts and change their keymap.

The goal of application templates is to support switching to a customized configuration without disrupting your existing settings and installation. This means people can build their own *applications* on top of Blender that can be easily distributed.

Details

An application template may define its own:

Startup File
    The default file to load with this template.

Preferences

Only certain preferences from a template are used:

- Themes.
- Add-ons.
- Keymaps.
- Viewport lighting.

Splash Screen

Templates may provide their own splash screen image.

Python Scripts

While templates have access to the same functionality as any other scripts, typical operations include:

- Modifying and replacing parts of the user interface.
- Defining new menus, keymaps and tools.
- Defining a custom add-on path for template specific add-ons.

Templates also have their own user configuration, so saving a startup file while using a template won't overwrite your default startup file.

Directory Layout

Templates may be located in one of two locations within the `scripts` directory.

Template locations:

```
{BLENDER_USER_SCRIPTS}/startup/bl_app_templates_user
{BLENDER_SYSTEM_SCRIPTS}/startup/bl_app_templates_system
```

User configuration is stored in a subdirectory:

Without a template:

```
./config/startup.blend
./config/userpref.blend
```

With a template:

```
./config/{APP_TEMPLATE_ID}/startup.blend
./config/{APP_TEMPLATE_ID}/userpref.blend
```

See Blender’s Directory Layout for details on script and configuration locations.

Hint

Troubleshooting Paths

When creating an application template, you may run into issues where paths are not being found. To investigate this you can log output of all of Blender’s path look-ups.

Example command line arguments that load Blender with a custom application template (replace `my_app_template` with the name of your own template):

```
blender --log "bke.appdir.*" --log-level -1 --app-template
my_app_template
```

You can then check the paths where attempts to access `my_app_template` are made.

Command Line Access

Using the command-line arguments you can setup a launcher that opens Blender with a specific app template:

```
blender --app-template my_template
```

Template Contents

Each of the following files can be used for application templates but are optional.

`startup.blend`

Factory startup file to use for this template.

`userpref.blend`

Factory preferences file to use for this template. When omitted preferences are shared with the default Blender configuration.

(As noted previously, this is only used for a subset of preferences).

`splash.png`

Splash screen to override Blender's default artwork (not including header text). Note, this image must be a `1000x500` image.

`__init__.py`

A Python script which must contain `register` and `unregister` functions.

Note

Bundled blend-files `startup.blend` and `userpref.blend` are considered *Factory Settings* and are never overwritten.

The user may save their own startup/preferences while using this template which will be stored in their user configuration, but only when the template includes its own `userpref.blend` file.

The original template settings can be loaded using: *Load Template Factory Settings* from the file menu in much the same way *Load Factory Settings* works.

Template Scripts

While app templates can use Python scripts, they simply have access to the same APIs available for add-ons and any other scripts.

As noted above, you may optionally have an `__init__.py` in your app template. This has the following advantages:

- Changes can be made to the startup or preferences, without having to distribute a blend-file.
- Changes can be made dynamically.

  You could for example – configure the template to check the number of processors, operating system and memory, then set values based on this.
- You may enable add-ons associated with your template.

On activation a `register` function is called, `unregister` is called when another template is selected.

As these only run once, any changes to defaults must be made via handler. Two handlers you are likely to use are:

- `bpy.app.handlers.load_factory_preferences_post`
- `bpy.app.handlers.load_factory_startup_post`

These allow you to define your own “factory settings”, which the user may change, just as Blender has it’s own defaults when first launched.

This is an example `__init__.py` file which defines defaults for an app template to use.

```
import bpy
from bpy.app.handlers import persistent

@persistent
def load_handler_for_preferences(_):
    print("Changing Preference Defaults!")
    from bpy import context
    prefs = context.preferences
    prefs.use_preferences_save = False

    kc = context.window_manager.keyconfigs["blender"]
    kc_prefs = kc.preferences
    if kc_prefs is not None:
        kc_prefs.select_mouse = 'RIGHT'
        kc_prefs.spacebar_action = 'SEARCH'
        kc_prefs.use_pie_click_drag = True

    view = prefs.view
    view.header_align = 'BOTTOM'


@persistent
def load_handler_for_startup(_):
    print("Changing Startup Defaults!")

    # Use smooth faces.
    for mesh in bpy.data.meshes:
        for poly in mesh.polygons:
            poly.use_smooth = True

    # Use material preview shading.
    for screen in bpy.data.screens:
        for area in screen.areas:
            for space in area.spaces:
                if space.type == 'VIEW_3D':
                    space.shading.type = 'MATERIAL'
                    space.shading.use_scene_lights = True


def register():
    print("Registering to Change Defaults")

bpy.app.handlers.load_factory_preferences_post.append(load_hand
ler_for_preferences)

bpy.app.handlers.load_factory_startup_post.append(load_handler_
for_startup)

def unregister():
    print("Unregistering to Change Defaults")
```

```
bpy.app.handlers.load_factory_preferences_post.remove(load_handler_for_preferences)

bpy.app.handlers.load_factory_startup_post.remove(load_handler_for_startup)
```

Keymap Customization

Keys

Available Keys

When customizing keymaps it's useful to use keys which won't conflict with Blender's default keymap.

Here are keys which aren't used and aren't likely to be used in the future.

F-Keys (`F5` - `F8`)

These F-keys (including modifier combination) have been intentionally kept free for users to bind their own keys to.

`OSKey` (also known as the `Windows-Key`, `Cmd` or `Super`)

Blender doesn't use this key for any bindings.

macOS is an exception, where `Cmd` replaces `Ctrl` except in cases it would conflict with the system's key bindings.

Modifier Double Click

Binding modifier keys as primary keys is supported, to avoid conflicts with regular usage you can bind them to double click.

Multi-Action Keys

Click/Drag

It's possible to configure a single key to perform multiple operations using *Click* event instead of *Press*. Then you may bind *Drag* to a separate action.

This is useful for mixing actions where one uses a drag event, e.g: Toggle a setting using with `Tab`, drag to open a pie menu showing all options related to the setting.

This is used in the default keymap in the 3D Viewport, `Alt-MMB` dragging in different directions rotates the view.

Common Operations

This section lists useful generic operations which can be used.

Key Bindings for Pop-Ups

Menus and panels can be assigned key shortcuts, even if they're only accessible from submenus elsewhere.

Open a Pop-up Menu (`wm.call_menu`)
    Open any menu on key press.

Open a Pie Menu (`wm.call_menu_pie`)
    Open any pie menu on key press.

Open a Panel (`wm.call_panel`)
    Open a pop-up panel (also known as a pop-over).

Menu & Panel Identifiers

To find the `name` of a menu, enable the preference *Interface ‣ Display ‣ Python Tooltips*.

Then hover the cursor over the popover button or menu item. For submenus you will need to use the back arrow to prevent the submenu from opening and gaining focus.

Key Bindings for Properties

There are many properties you might want to bind a key with. To avoid having to define operators for each property, there are generic operators for this purpose:

Operators for adjusting properties begin with `wm.context_`.

Some of these include:

- `wm.context_toggle` toggle a Boolean property.
- `wm.context_cycle_enum` cycle an enum property forwards or backwards.
- `wm.context_menu_enum` show a pop-up menu for an enum property.
- `wm.context_pie_enum` show a pie menu for an enum property.
- `wm.context_scale_float` scale a number (used for increasing / decreasing brush size for example).
- `wm.context_toggle_enum` toggle between two options of an enum.
- `wm.context_modal_mouse` moving the cursor to interactively change a value.

See bpy.ops.wm for a complete list.

Each of these operators has a `data_path` setting to reference the property to change.

To find the `data_path`, basic Python knowledge is needed.

For example, you can use the Python Console to access a Boolean property you wish to map to a key:

```
bpy.context.object.show_name
```

To bind this to a key, add a new keymap item using the operator `wm.context_toggle` with `data_path` set to `object.show_name` (notice the `bpy.context` prefix is implicit).

See bpy.context for other context attributes.

The Python API documentation can be used to find properties or you may use the Python Console's auto-complete to inspect available properties.

Working Limits

Space

While object positions, vertex locations are not clamped, larger values become increasingly imprecise. To get an idea of the precision you can work with using different scales. Here’s a table of scales and their associated accuracy:

10::	1/1,048,576th
100::	1/131,072th
1,000::	1/16,384th
10,000::	1/1,024th
100,000::	1/128th
1,000,000::	1/16th

> **Hint**
>
> For a rough rule of thumb, values within -5,000/+5,000 are typically reliable (range of 10,000). Internally *single precision* floating-point calculations are used.

Time

The maximum number of frames for each scene is currently 1,048,574, and allows for continuous shots for durations of:

24 fps::	12 hours, 8 minutes.
25 fps::	11 hours, 39 minutes.
30 fps::	9 hours, 42 minutes.
60 fps::	4 hours, 51 minutes.

Note

In practice, a finished work is typically composed of output from many scenes. So this limit does not prevent you from creating longer works.

Text Fields

Fixed strings are used internally, and while it is not useful to list all limits, here are some common limits. *Used for data-block names, modifiers, vertex groups, UV layers…*

directory::	767
file-name::	255
file-path::	1023
identifier::	63

Note

Multi-byte encoding means some Unicode characters use more than a single ASCII character.

Operators

Operator Cheat Sheet

Reference

Menu::	*Help* ▸ *Operator Cheat Sheet*
Context::	Enable Developer Extras

Creates a text file in the Text Editor that gives a list of all operators and their default values in Python syntax, along with the generated docs. This is a good way to get an overview of all Blender's operators.

See also

Blender's API documentation

System Operators

Reload Scripts

Reference

Mode::	All Modes
Menu::	*Topbar* ▸ *Blender* ▸ *System* ▸ *Reload Scripts*

Reloads all scripts found in the scripts data folder; any changes that have been made in the Text Editor will be lost!

Memory Statistics

Reference

Mode::	`--debug-memory`
Menu::	*Topbar ‣ Blender ‣ System ‣ Memory Statistics*

This operator which can be found by searching "Memory Statistics" with the Operator Search will print useful information about memory objects, their size and user count.

Important

To fully use this operator run Blender from the console with `--debug-memory`.

Debug Menu

Reference

Mode::	All Modes
Menu::	*Topbar ‣ Blender ‣ System ‣ Debug Menu*

This operator brings up a menu to set Blender into a certain debug mode.

See the source code for a description of what each value does.

Tip

Developers can search the code for `G.debug_value` to find other possible uses for this operator.

Note

Additional debug options are available by launching Blender in debug mode or setting `bpy.app.debug = True`.

Redraw Timer

Reference

Mode:: All Modes
Menu:: *Topbar ‣ Blender ‣ System ‣ Redraw Timer*

This operator brings up a menu with a list of tests to benchmark UI render times along with undo/redo functions.

Clean Up Space-Data

Reference

Mode:: All Modes
Menu:: *Topbar ‣ Blender ‣ System ‣ Clean Up Space-data*

Removes unused settings for invisible editors.

Blender’s Directory Layout

This page documents the different directories used by Blender.

This can be helpful for troubleshooting, automation and customization.

User Directories

User directories store preferences, startup file, installed extensions, presets and more. By default these use the standard configuration folders for each operating system.

Linux

```
$HOME/.config/blender/4.3/
```

If the `$XDG_CONFIG_HOME` environment variable is set:

```
$XDG_CONFIG_HOME/blender/4.3/
```

macOS

```
/Users/$USER/Library/Application Support/Blender/4.3/
```

Windows

```
%USERPROFILE%\AppData\Roaming\Blender Foundation\Blender\4.3\
```

Portable Installation

When running Blender from a portable drive, it’s possible to keep the configuration files on the same drive to take with you.

To enable this, create a folder named `portable` at the following locations:

- Windows: Next to the Blender executable, in the unzipped folder
- Linux: Next to the Blender executable, in the unzipped folder
- macOS: Inside the application bundle at `Blender.app/Contents/Resources`

This folder will then store preferences, startup file, installed extensions and presets.

Environment Variables

The `BLENDER_USER_RESOURCES` environment variable can be set to a custom directory to replace the default user directory.

System Directories

System directories store files that come bundled with Blender and are required for it to function. This includes scripts, presets, essential assets and more.

Linux

Archive downloaded from blender.org:

```
./4.3/
```

Linux distribution packages:

```
/usr/share/blender/4.3/
```

macOS

```
./Blender.app/Contents/Resources/4.3/
```

Windows

Zip file downloaded from blender.org:

`./4.3/`

Installer downloaded from blender.org:

`%ProgramFiles%\Blender Foundation\Blender\4.3\`

Microsoft Store installation:

`%ProgramFiles%\WindowsApps\BlenderFoundation.Blender<HASH>\Blender\4.3\`

Environment Variables

`BLENDER_SYSTEM_SCRIPTS` and `BLENDER_SYSTEM_EXTENSIONS` environment variables can be used to bundle additional scripts and extensions, that are not part of the regular Blender installation.

Other `BLENDER_SYSTEM` environment variables can override other system paths, though are not commonly used in practice.

Path Layout

`./autosave`

Autosave blend-file location. (Windows only, temp directory used for other systems.)

Located in user directories.

`./config`

User configuration and session info.

Located in user directories.

`./config/startup.blend`

Blend file to load on startup.

`./config/userpref.blend`

User preferences.

`./config/bookmarks.txt`
: File Browser bookmarks.

`./config/recent-files.txt`
: Recent file menu list.

`./config/{APP_TEMPLATE_ID}/startup.blend`
: Startup file for an application template.

`./config/{APP_TEMPLATE_ID}/userpref.blend`
: User preferences file for an application template.

`./datafiles`

Data files loaded at runtime.

Located in both user and system directories. User data files either override or add to system data files.

`./datafiles/colormanagement`
: Default OpenColorIO configuration.

`./datafiles/fonts`
: User interface fonts.

`./datafiles/studiolights`
: Studio light images for 3D viewport.

`./extensions`

Extension repositories.

Located in both user and system directories. Repositories are loaded from both directories.

`./scripts`

Add-ons, presets, templates, user interface, startup scripts.

Located in both user and system directories. Scripts are loaded from both directories.

`./scripts/addons/*.py`

Python add-ons which may be enabled in the Preferences include import/export format support, render engine integration and many handy utilities.

`./scripts/addons/modules/*.py`

Modules for add-ons to use (added to Python's `sys.path`).

`./scripts/addons_core/*.py`

The add-ons directory which is used for bundled add-ons.

`./scripts/addons_core/modules/*.py`

Modules for `addons_core` to use (added to Python's `sys.path` when it found).

`./scripts/modules/*.py`

Python modules containing our core API and utility functions for other scripts to import (added to Python's `sys.path`).

`./scripts/startup/*.py`

Scripts which are automatically imported on startup.

  `./scripts/startup/bl_app_templates_user/{APP_TEMPLATE_ID}`

  Application templates installed in user directories.

  `./scripts/startup/bl_app_templates_system/{APP_TEMPLATE_ID}`

  pplication templates automatically loaded from system directories.

`./scripts/presets/{preset}/*.py`

Presets used for storing user-defined settings for cloth, render formats, etc.

`./scripts/templates_py/*.py`

Example scripts which can be accessed from *Text Editor* ‣ *Templates* ‣ *Python*.

`./scripts/templates_osl/*.osl`

Example OSL shaders which can be accessed from *Text Editor* ‣ *Templates* ‣ *Open Shading Language*.

`./python`

Bundled Python distribution.

Located in system directories.

Local Cache Directory

The cache directory is used to store persistent caches locally. Currently it is only used for the indexing of Asset Libraries. The operating system is not expected to clear this automatically.

The following path will be used:

- **Linux::**	`$XDG_CACHE_HOME/blender/` if `$XDG_CACHE_HOME` is set, otherwise `$HOME/.cache/blender/`
- **macOS::**	`/Library/Caches/Blender/`
- **Windows::**	`%USERPROFILE%\AppData\Local\Blender Foundation\Blender\Cache\`

Temporary Directory

The temporary directory is used to store various files at run-time (including render layers, physics cache, copy-paste buffer and crash logs).

The temporary directory is selected based on the following priority:

- User Preference (see File Paths).
- Environment variables (`TEMP` on Windows, `TMP` & `TMP_DIR` on other platforms).
- The `/tmp/` directory.

Deploying Blender in Production

This page contains tips for setting up Blender in environments such as animation studios and schools.

These environments often have special requirements regarding security, automated deployment and customization.

Installing Blender

Blender downloads can be extracted to any directory on the system, as a self contained installation. Multiple Blender versions can co-exist on the same system, and deployment can be automated using standard file management tools.

New Blender versions may add, remove or change functionality that affects the results of production files. For a given project, it is advisable to use a single LTS version of Blender. LTS versions receive bug fixes for two years.

Working Offline

For security or other reasons, workstation may not have internet access.

By default Blender does not access the internet, however this can be enabled in the System preferences with the Online Access option.

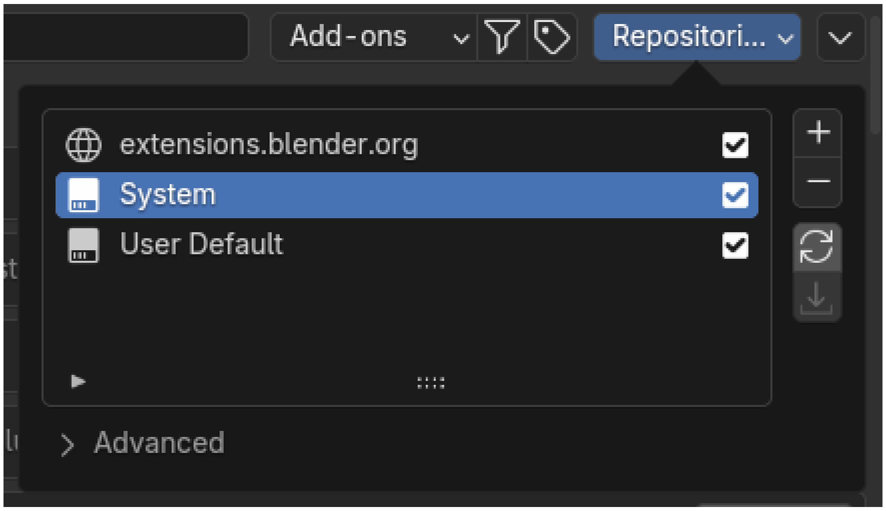
Working offline can be enforced by running with the `--offline-mode` command line argument. Users will then be unable to enable online access in the preferences.

Note

Add-ons that follow this setting will only connect to the internet if enabled. However, Blender cannot prevent third-party add-ons from violating this rule.

Bundling Extensions

When working offline or in a more controlled environment, it may be useful to provide a set of extensions to all users. For this there is a default read-only System repository. This repository can for example be located on a read-only network drive or in a system directory.



System repository

The `$BLENDER_SYSTEM_EXTENSIONS` environment variable controls the default location. This should point to a directory, within which a `system` directory should exist.

Extensions packages should be extracted in this `system` directory, with a resulting path like this:

```
$BLENDER_SYSTEM_EXTENSIONS/system/my-
addon/blender_manifest.toml
```

In the Extensions preferences, it’s possible to manually set a custom directory for the default System repository, or to create multiple repositories.

Bundling Scripts

Besides extensions, it’s possible to bundle scripts for presets, application templates, legacy add-ons, as well as scripts run on startup.

Script directories can be manually added in the File Paths preferences. The `$BLENDER_SYSTEM_SCRIPTS` can also be used to add a script directory without modifying the preferences.

These script directories are expected to contain specific directories like `presets`, `addons` and `startup` for different types of scripts. See Path Layout for a complete list.

Startup Scripts

The Blender Python API can be used to customize Blender. This includes changing preferences, changing the startup file and adding UI elements.

For example, a script can enable add-ons for every user.

`$BLENDER_SYSTEM_SCRIPTS/startup/enable_addons.py`

```
def register():
    import addon_utils
    addon_utils.enable("my-addon")

def unregister():
    pass

if __name__ == "__main__":
    register()
```

Application Templates

Application Templates can be used to set up Blender for particular tasks or projects, separate from the default configuration. When creating a new file the user can choose the template.

The files are expected to be placed in the system script directories like this:

```
$BLENDER_SYSTEM_SCRIPTS/startup/bl_app_templates_system/MyTempl
ate/__init__.py
$BLENDER_SYSTEM_SCRIPTS/startup/bl_app_templates_system/MyTempl
ate/startup.blend
```

Legacy Add-ons

Add-ons that have not been converted to become an extension yet need to be placed in the `addons` script directory.

For example, an add-on could be located at:

```
$BLENDER_SYSTEM_SCRIPTS/addons/simple_addon.py
$BLENDER_SYSTEM_SCRIPTS/addons/complex_addon/__init__.py
```

VFX Platform

Blender follows the VFX reference platform, which means it is able to run on the same systems as other VFX software and exchange image, volume and scene files with them.

Python Version

Blender and the bpy module are only compatible with a single Python version. This makes it possible for add-ons and VFX software in general to only have to target a single Python version.

Blender bundles a complete Python installation and does not interact with the system Python by default. This can be changed with the `--python-use-system-env` command line argument, if care is taken to set up a compatible Python version.

Appendices

This chapter covers far more detailed explanations about some Blender tools (which may not be required for typical usage).

- Rotation Modes

Rotation Modes

Blender lets you define rotations in several ways. Each one of them has a series of advantages and drawbacks; there is no best rotation mode, as each one is suitable for specific cases.

In all of these modes, positive angle values mean counter-clockwise rotation direction, while negative values define clockwise rotation.

Though you can rotate elements using the global or local transform orientations, these axes are not suitable to define rotations, as the effect of each of them cannot be isolated from the other two.

Take, for instance, any three values for X, Y and Z rotation. Perform each one of these using global or local axes. Depending on the order in which you perform these, you will end up with different final orientations. So proper rotation coordinate systems are needed.

Euler Modes

The axes system used for performing Euler rotations is the so called Euler gimbal. A gimbal is a particular set of three axes. The special thing about this is that the axes have a hierarchical relationship between them: one of the axes is at the top of the hierarchy, and has one of the other two axes as its immediate child; at the same time, this child axis is the parent of the remaining axis, the one at the very bottom of the hierarchy.

Which axis is on top, which one in the middle and which at the bottom, depends on the particular Euler gimbal: there are six types of them, as there are six possible combinations: XYZ, XZY, YXZ, YZX, ZXY and ZYX Euler rotation modes. These modes are named using the letters of the axes in order, starting from the axis at the bottom of the hierarchy, and finishing with the one on top.

The main problem of these systems comes when they lose their relative perpendicularity. And this happens when the axis in the middle rotates, causing the axis at the bottom to rotate with it. It keeps getting worse when this bottom axis approaches 90° (or equivalent angles). In that case, it will remain aligned with the axis on top of the hierarchy. In that moment we have just lost one axis of rotation. This can cause discontinuous interpolations when animating. This particular loss of axis is known as the "gimbal lock".

Hint

The actual configuration of the gimbal axes can be seen in the 3D Viewport by enabling the *Rotate* object gizmo and setting it to *Gimbal* (from the gizmos button in the header). At the same time, rotation mode should be set to any of the Euler modes for the active object.

Now you can perform a rotation around the axis in the middle (e.g. in *XYZ Euler* mode that is the Y axis), and see how easy it is to end up having a gimbal with just two axes. In the specific case of the *XYZ Euler* mode with gimbal lock, a rotation around the X axis will have the same effect as rotating around the Z axis, meaning, in practice, that no X axis rotations can be performed.

One advantage of this mode is that animation curves are easy to understand and edit. However, special attention must be done when the middle axis approaches values close to 90° (or equivalent angles).

Axis Angle Mode

This mode lets us define an axis (X, Y, Z) and a rotation angle (W) around that axis.

If we define the rotation using interactive rotations (with the rotation gizmo), the values of X, Y and Z will not exceed 1.0 in absolute value, and W will be comprised between 0 and 180 degrees.

If you wish to define rotations above 180° (e.g. to define multiple revolutions), you will need to edit the W value directly, but as soon as you perform an interactive rotation, that value will be adjusted again. Same thing goes for axis values.

This system is suitable for elements revolving around a fixed axis, or to animate one of the elements at a time (either the axis or the angle). The problem might come when animating (interpolating) both components at the same time: axis and angle. The resulting effect might not be as expected.

The *Gimbal* gizmo in this rotation mode shows a set of three orthogonal axes in which the Z axis goes along the defined rotation axis, i.e. it points towards the direction defined by the (X, Y, Z) point.

The axis-angle system is free from gimbal lock, but animation curves in this mode are not intuitive at all when animating axis and angle at the same time, in which case they are difficult to understand and edit.

Quaternion Mode

In this mode, rotations are also defined by four values (X, Y, Z and W). X, Y and Z also define an axis, and W an angle, but it does it quite differently from axis-angle. The important thing here is the relation between all four values.

To describe it in an intuitive way, let's take the effect of the X coordinate: what it does is to rotate the element around the X axis up to 180 degrees. The same goes for Y and Z. The effect of W is to avoid those rotations and leave the element with zero rotation. The final orientation is a combination of these four effects.

As the relation between components is what defines the final orientation, multiplying or dividing all four numbers by a constant value will yield the very same rotation.

This mode is ideal for interpolating between **any** pair of orientations. It doesn't suffer from gimbal lock or any interpolation undesired effect. The only drawback is that you cannot interpolate between two orientations that are at a distance greater than than 180°, as the animation will take the shortest path between them. Thus to animate a revolving element you must set up many intermediate keyframes, 180° from each other at most.

The *Gimbal* gizmo in this mode is equivalent to the *Local* one, and doesn't have any special meaning.

The animation curves in this mode are not intuitive, so they are also difficult to understand and edit.

More about Quaternions

This section is not really useful for 3D artists, but it can be suitable for the curious or the scientist.

Quaternions are a number system extending the complex numbers. They represent a four component vector, whose components are called, in Blender, X, Y, Z and W. When rotating interactively in quaternion mode, the so called norm (length) of the quaternion will remain constant. By definition, the norm of a quaternion equals 1.0 (that's a **normalized** quaternion). When you select the quaternion mode in Blender, the XYZW components describe a normalized quaternion.

> **Note**
>
> The norm of a quaternion *q* is defined mathematically as:
>
> $$\lvert q \rvert = \sqrt{X^2 + Y^2 + Z^2 + W^2}$$

However, if one of the quaternion components is locked during the interactive transformation using the proper lock button, the norm will not remain unchanged, as that blocked component will not be able to adjust itself to keep the unit norm.

Hint

Interactive rotations with the gizmo don’t change the norm of the current quaternion. Editing a single XYZW component individually you can change the norm. To make the norm 1.0 again you can switch to any rotation mode and back again into quaternion.

The rotation components of a quaternion keep a tight relation with those of axis-angle. To find a correspondence, first of all we must deal with the normalized version of the quaternion, that is, one whose norm equals 1.0. To normalize a quaternion, just divide each one of its components by its norm. As we have seen before, dividing all four values by the same number gives the same orientation.

Once we have calculated the components of the normalized quaternion, the relation with the axis-angle components is as follows:

- X, Y and Z mean exactly the same as in axis-angle: they just define an axis around which the rotation takes place.
- W can be used to retrieve the actual rotation around the defined angle. The following formula applies (provided that the *quaternion is normalized*): $W = \cos(\frac{a}{2})$, where *a* is actually the rotation angle we are looking for. That is: $a = 2 \arccos{W}$.

Other Considerations

In axis-angle and quaternion modes we can lock rotations in interactive modes in a per component basis, instead of doing it by axis. To do so we can activate this locking ability using the lock buttons next to the corresponding *Rotation* transform buttons.

Regarding rotation animations, all keyframes must be defined in the same rotation mode, which must be the selected rotation mode for the object throughout the entire animation.

Troubleshooting

- Startup
- 3D Viewport
- Graphics Hardware
- Crashes
- Python Errors
- Recovering Data

Compatibility

Some applications which integrate themselves into your system can cause problem's with Blender.

Here is a list of known compatibility issues.

Startup

There are some common causes for problems when using Blender. If Blender fails to start, there are a few things to check for:

- See if your computer meets the minimum requirements.
- Confirm that your graphics card is supported and that the drivers are up to date (see Troubleshooting Graphics Hardware).
- Make sure any antivirus software is not preventing Blender from starting.

If you cannot find a solution to your problem here, try asking the community for help.

Common Startup Messages

The *Blender Console Window* can display many different types of status and error messages. Some messages simply inform the user what Blender is doing, but have no real impact on Blender's ability to function. Other messages can indicate serious errors that will most likely prevent Blender carrying out a particular task and may even make Blender non-responsive or shut down completely. The *Blender Console Window* messages can also originate internally from within the Blender code or from external sources such as Python scripts.

`found bundled python: {DIR}`

This message indicates that Blender was able to find the Python library for the Python interpreter embedded within Blender. If this folder is missing or unable to be found, it is likely that an error will occur, and this message will not appear.

`Read prefs: {DIR}/userpref.blend`

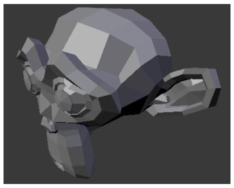
The preferences use this path.

3D Viewport

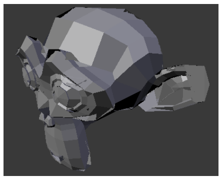
Rendering

Depth Buffer Glitches

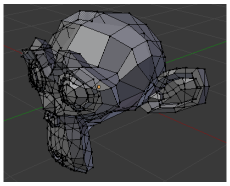
Sometimes when setting a large clipping range will allow you to see both near and far objects, but reduces the depth precision resulting in artifacts.



Model with no clipping artifacts.



Model with clipping artifacts.



Mesh with artifacts in Edit Mode.

To avoid this:

- Increase the near clipping when working on large scenes.
- Decrease the far clipping when objects are not viewed at a distance.

When perspective is disabled only the far Clip End is used, very high values can still result in artifacts. This is **not** specific to Blender, all graphics applications have these same limitations.

Objects Invisible in Camera View

If you have a large scene, viewing it through Camera View may not display all of the objects in the scene. One possibility may be that the clipping distance of the camera is too low. The camera will only show objects that fall within the clipping range.

Performance

Slow Rendering

There are a couple of reasons why you may be experiencing a slow viewport.

Old Hardware

Sometimes your hardware, mainly your graphics card, may be too slow to keep up with your model.

Upgrade Graphics Driver

In some cases, slow selection is resolved by using updated drivers.

Slow Selection

Blender uses OpenGL for selection, some graphics card drivers are slow at performing this operation.

This becomes especially problematic on dense geometry.

Possible Solutions:

GPU Depth Picking (Preferences)

See *Preferences ‣ Viewport ‣ Selection*.

This option is enabled by default, disabling it may give a better performance at the cost of selection accuracy.

Upgrade Graphics Driver

In some cases, slow selection is resolved by using updated drivers. *It is generally good to use recent drivers when using 3D software*.

Select Centers (Workaround)

In *Object Mode*, holding `Ctrl` while selecting uses the object center point. While this can be useful on its own, it has the side effect of not relying on OpenGL selection.

Change Display Mode (Workaround)

Using *Wireframe* display mode can be used to more quickly select different objects.

Note

Obviously, the workarounds listed here are not long term solutions, but it is handy to know if you are stuck using a system with poor OpenGL support.

Ultimately, if none of these options work out it may be worth upgrading your hardware.

Viewport Playback Frame Rate Limited

Having the viewport playback clamped to a maximum of 60 FPS is typically caused by the VSYNC setting on your GPU, for higher frame rates you may have to disable VSYNC functionality although this may be of limited us since frames rendered may be more than your GPU and monitor are able to display.

VSYNC is configured as part of your GPU driver options which vary depending on your system & GPU combination.

Navigation

Lost in Space

When navigating your scene, you may accidentally navigate away from your scene and find yourself with a blank viewport. There are two ways to fix this:

1. Select an object in the Outliner, then zoom to that object with *View ‣ Frame Selected* or `NumpadPeriod`.
2. Use `Home` to fit all objects into the 3D Viewport.

Invisible Limit Zooming In

Sometimes when navigating you may be trying to zoom in but it seems that you have hit a limit to how far you can zoom. This is because Blender uses a central point to orbit around.

In practice this is good for modeling an object which you rotate about a lot to see from all sides (think of a potter using a wheel). However, this makes it awkward to explore a scene or model an object from the 'inside', for example.

Solutions

- Use View Dolly.
- Use Walk/Fly Navigation.
- Use Auto Depth and Zoom to Mouse Position. These tools will make sure the distance is always the value under the mouse cursor,
- Use Zoom Region as it also resets the center point when zooming.
- Center the view around the mouse cursor `Alt-MMB`. This will take the position under the cursor and make it your viewpoint center.
- Use an NDOF, also known as a 3D mouse, see configuring peripherals for more information.

Tools

Invalid Selection

There are times when selection fails under some configurations, often this is noticeable in mesh *Edit Mode*, selecting vertices/edges/faces where random elements are selected.

Internally Blender uses OpenGL for selection, so the graphics card driver relies on giving correct results.

Possible Solutions:

Disable Multisampling

This is by far the most common cause of selection issues.

There are known problems with some graphics cards when using multisampling.

You can disable this option by turning multisampling off in your graphics card driver options.

Change Anti-Aliasing Sample Settings

Depending on your OpenGL configuration, some specific sample settings may work while others fail.

Unfortunately finding working configuration involves trial & error testing.

Upgrade Graphics Driver

As with any OpenGL-related issues, using recent drivers can resolve problems.

However, it should be noted that this is a fairly common problem and remains unresolved with many drivers.

Troubleshooting Graphics Hardware

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

- Windows
    - Nvidia
    - AMD
    - Intel
    - Other GPU
- Linux
    - Nvidia
    - AMD
    - Intel
    - Other GPU
- macOS
    - Nvidia
    - AMD
    - Intel
    - Other GPU

Troubleshooting Windows Hardware

- Nvidia
  - Drivers
  - Laptops
  - Common Problems
  - Information
- AMD
  - Drivers
  - Laptops
  - Common Problems
  - Information
- Intel
  - Drivers
  - Laptops
  - Compatibility
  - Common Problems
  - Information
  - Legacy Intel HD 4000/5000
- Other GPU
  - Drivers
  - Laptops
  - Common Problems
  - Information

Windows – Nvidia

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Windows drivers are provided by the graphics card manufacturer (Nvidia). Windows update automatically installs graphics drivers, or your computer manufacturer may provide its own version of the graphics drivers.

However, these are not always the latest version or may have been corrupted in some way. We recommend to always use the official drivers.

Download Latest Nvidia Drivers

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ‣ *System* ‣ *Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Windows – AMD

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender’s behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Windows drivers are provided by the graphics card manufacturer (AMD). Windows update automatically installs graphics drivers, or your computer manufacturer may provide its own version of the graphics drivers.

However, these are not always the latest version or may have been corrupted in some way. We recommend to always use the official drivers.

Download Latest AMD Drivers

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences ‣ System ‣ Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Windows – Intel

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender’s behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Windows drivers are provided by the graphics card manufacturer (Intel). Windows update automatically installs graphics drivers, or your computer manufacturer may provide its own version of the graphics drivers.

However, these are not always the latest version or may have been corrupted in some way. We recommend to use the official drivers.

Download Latest Intel Drivers

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Compatibility

In some cases Blender may crash on startup. Running Blender in compatibility mode can help in fixing this issue. To enable compatibility mode, `RMB` on the Blender executable and select *Properties* ▸ *Compatibility* and enable *Run this program in compatibility mode*. Confirm the changes with *Apply*.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer's website.

Poor Performance

- Update your graphics drivers (see above).

- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences ‣ System ‣ Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help ‣ Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Legacy Intel HD 4000/5000

When running on Intel 3, 4 or 5th gen iGPU the latest Intel driver will crash on startup. In order to start Blender try to install a previous version of the driver. Drivers that are known to work are:

- 20.19.15.4835
- 20.19.15.4963
- 20.19.15.5063

Drivers that are known to fail are:

- 20.19.15.5144
- 20.19.15.5166
- 20.19.15.5171

Download older Intel Drivers

Windows – Other GPU

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Windows drivers are provided by the graphics card manufacturer. Windows update automatically installs graphics drivers, or your computer manufacturer may provide its own version of the graphics drivers.

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ▸ *System* ▸ *Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Troubleshooting Linux Hardware

- Nvidia
    - Drivers
    - Laptops
    - Common Problems
    - Information
- AMD
    - Drivers
    - Laptops
    - Common Problems
    - Information
- Intel
    - Drivers
    - Common Problems
    - Information
- Other GPU
    - Drivers
    - Laptops
    - Common Problems
    - Information

Linux – Nvidia

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Linux, graphics drivers are usually installed as a package by your Linux distribution. Installing the latest drivers is typically done by upgrading packages or the distribution as a whole. Some distributions provide multiple packages for multiple drivers versions, giving you the choice to install newer versions.

For Nvidia there are open source (Nouveau) and closed source (by Nvidia) graphics drivers. Blender functions best with the closed source drivers as they are more optimized and complete. Linux graphics drivers can be downloaded from Nvidia's website, however in most cases the ones from your Linux distribution are fine and make things easier. Manually downloading drivers is mostly useful to get the very latest version, for example for a GPU that was only recently released.

NVidia Website

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ▸ *System* ▸ *Memory & Limits*.

- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Linux – AMD

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Linux, graphics drivers are usually installed as a package by your Linux distribution. Installing the latest drivers is typically done by upgrading packages or the distribution as a whole. Some distributions provide multiple packages for multiple drivers versions, giving you the choice to install newer versions.

AMD drivers are open source, except for the OpenCL support which is available as part of Pro drivers. Installing packages through your Linux distribution is usually best. AMD also provides graphics drivers for download on their website if you need the latest version.

AMD Drivers and Support Website

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer's website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ▸ *System* ▸ *Memory & Limits*.

- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Linux – Intel

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Linux, graphics drivers are usually installed as a package by your Linux distribution. Installing the latest drivers is typically done by upgrading packages or the distribution as a whole. Some distributions provide multiple packages for multiple drivers versions, giving you the choice to install newer versions.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ‣ *System* ‣ *Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ‣ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Linux – Other GPU

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On Linux, graphics drivers are usually installed as a package by your Linux distribution. Installing the latest drivers is typically done by upgrading packages or the distribution as a whole. Some distributions provide multiple packages for multiple drivers versions, giving you the choice to install newer versions.

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer's website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ‣ *System* ‣ *Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Troubleshooting macOS Hardware

- Nvidia
    - Drivers
    - Common Problems
    - Information
- AMD
    - Drivers
    - Common Problems
    - Information
- Intel
    - Drivers
    - Common Problems
    - Information
- Other GPU
    - Drivers
    - Common Problems
    - Information

macOS – Nvidia

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On macOS graphics drivers are built into the operating system and the only way to get newer drivers is to upgrade macOS as a whole to the latest version.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer's website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences ‣ System ‣ Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help ‣ Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

macOS – AMD

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On macOS graphics drivers are built into the operating system and the only way to get newer drivers is to upgrade macOS as a whole to the latest version.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences ‣ System ‣ Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help ‣ Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

macOS – Intel

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On macOS graphics drivers are built into the operating system and the only way to get newer drivers is to upgrade macOS as a whole to the latest version.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer's website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ▸ *System* ▸ *Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help* ▸ *Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

macOS – Other GPU

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender's behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

On macOS graphics drivers are built into the operating system and the only way to get newer drivers is to upgrade macOS as a whole to the latest version.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer’s website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences ‣ System ‣ Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help ‣ Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.

Crashes

The most common causes of Blender crashes:

- Running out of memory.
- Issues with graphics hardware or drivers.
- Bugs in Blender.

Firstly, you may be able to recover your work with *File ‣ Recover ‣ Auto Save…*.

To prevent the problem from happening again, you can check that the graphics drivers are up to date (Troubleshooting Graphics Hardware), upgrade your machine's hardware (the RAM or graphics card), and disable some options that are more memory intensive:

- Reduce undo steps *Preferences ‣ System ‣ Memory & Limits ‣ Undo Steps*.
- Using multisample anti-aliasing also increases the memory usage and makes the display slower.
- On Linux, the Window Manager (KDE and Gnome for example) may be using hardware accelerated effects (e.g. window shadows and transparency) that are using up the memory that Blender needs. Try disabling the desktop effects or switch to a lightweight Window Manager.

To check memory usage by Blender:

- On Windows, use Task Manager and sort by Memory.
- On macOS, use Activity Monitor.app and open Memory tab. Alternatively, run `top -o MEM`.
- On Linux, run `top -o %MEM`.

Crash Log

When Blender crashes, it writes out a text file which contains information that may help identify the cause of the crash. Usually, this file is written in the Temporary Directory directory.

This file contains a log of tools used up until the crash as well as some other debug information. When reporting bugs about crashes it can be helpful to attach this file to your reports, especially when others are unable to reproduce the crash.

Windows

On a crash, a file is written based on the name of the currently loaded blend-file, so `test.blend` will create a file called `test.crash.txt`.

Batch scripts are provided in Blender installation directory which may be run to obtain the Blender debug log and system info text files:

- `blender_debug_log.cmd` is used in most cases.
- `blender_debug_gpu.cmd` and `blender_debug_gpu_workaround.cmd` log GPU-related errors.
- `blender_factory_startup.cmd` starts Blender with default settings which is recommended for debugging.

If the crash happens in Blender module, stack trace is also written to a file named `blender.crash.txt`. The path to that file can be found at the end of `blender_debug_log.txt` file.

macOS

After crash, macOS Crash Reporter shows a window with backtrace after some time, or when Blender is opened again. Copy the text in the crash report and save it in a text file. That file should be attached to the bug report while following other bug reporting guidelines.

Some `.crash` files can also be found in `~/Library/Logs/DiagnosticReports/` with the name of format: `Blender_YYYY-MM-DD-HHMMSS_MACNAME.crash`. If a report is present corresponding to the time of crash, that file can also provide hints about cause of the crash. Alternatively, Console.app can be used to navigate all “User Reports” (see sidebar in the app).

Linux

On a crash, a file named `blender.crash.txt` is written in `/tmp` directory.

Note

More logs can be obtained by running Blender from Command Line and using `--factory-startup --debug-all` flags. See Launching from the Command Line and Command Line Arguments.

Python Errors

Precompiled Libraries

While not common practice, Python add-ons can be distributed with their own precompiled libraries. Unlike regular Python scripts, these are not portable between different platforms.

It is possible the library is incompatible with your Blender installation (attempting to load a library built for a different version of Python, or loading a 32-bit library on a 64-bit system).

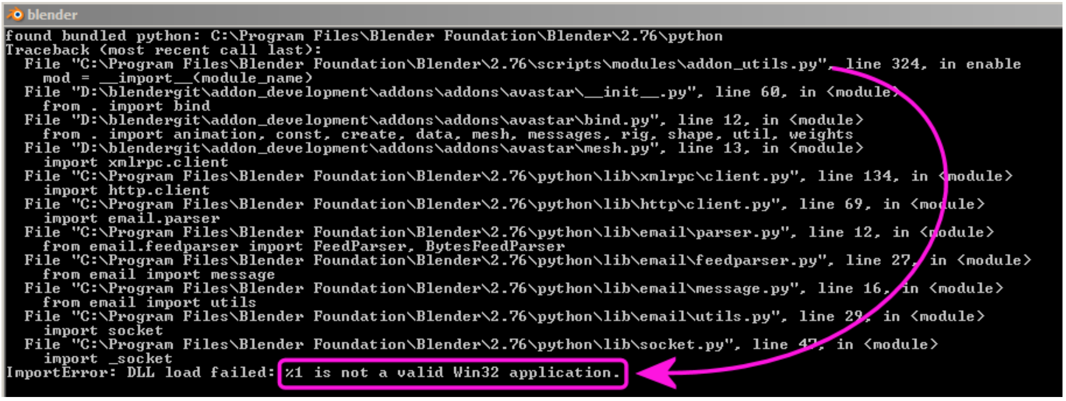
If the add-on contains `.pyd` or `.so` files, check that the distribution is compatible with your operating system.

Platform Specific

Windows

Mixed Python Libraries (DLLs)

If Python is raising errors or you have an add-on that just fails when enabled with an error, e.g: `... is not a valid Win32 application.`.



```
found bundled python: C:\Program Files\Blender Foundation\Blender\2.76\python
Traceback (most recent call last):
  File "C:\Program Files\Blender Foundation\Blender\2.76\scripts\modules\addon_utils.py", line 324, in enable
    mod = __import__(module_name)
  File "D:\blendergit\addon_development\addons\addons\avastar\__init__.py", line 60, in <module>
    from . import bind
  File "D:\blendergit\addon_development\addons\addons\avastar\bind.py", line 12, in <module>
    from . import animation, const, create, data, mesh, messages, rig, shape, util, weights
  File "D:\blendergit\addon_development\addons\addons\avastar\mesh.py", line 13, in <module>
    import xmlrpc.client
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\xmlrpc\client.py", line 134, in <module>
    import http.client
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\http\client.py", line 69, in <module>
    import email.parser
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\email\parser.py", line 12, in <module>
    from email.feedparser import FeedParser, BytesFeedParser
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\email\feedparser.py", line 27, in <module>
    from email import message
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\email\message.py", line 16, in <module>
    from email import utils
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\email\utils.py", line 29, in <module>
    import socket
  File "C:\Program Files\Blender Foundation\Blender\2.76\python\lib\socket.py", line 47, in <module>
    import _socket
ImportError: DLL load failed: %1 is not a valid Win32 application.
```

A Python traceback.

This may be caused by some inconsistency in the Python libraries. While Blender comes with its own bundled Python interpreter, duplicate, incompatible libraries can cause problems.

To find out which Python Library caused the Problem check the error message.

This is normally reported somewhere around the bottom line of the traceback. With the error above you see the problem is caused while trying to import `_socket`. This corresponds to either a file named `_socket.py` or `_socket.pyd`.

To help troubleshoot this problem, the following script can be pasted into the Text editor and run to check for duplicate libraries in your search path. (The output will show in Command Line Window.)

```
import os
import sys

# Change this based on the library you wish to test
test_lib = "_socket.pyd"

def GetSystemDirectory():
    from ctypes import windll, create_string_buffer, sizeof
    GetSystemDirectory = windll.kernel32.GetSystemDirectoryA
    buffer = create_string_buffer(260)
    GetSystemDirectory(buffer, sizeof(buffer))
    return os.fsdecode(buffer.value)

def library_search_paths():
    return (
        # Windows search paths
        os.path.dirname(sys.argv[0]),
        os.getcwd(),
        GetSystemDirectory(),
        os.environ["WINDIR"],  # GetWindowsDirectory
        *os.environ["PATH"].split(";"),

        # regular Python search paths
        *sys.path,
        )

def check_library_duplicate(libname):
    paths = [p for p in library_search_paths()
             if os.path.exists(os.path.join(p, libname))]

    print("Library %r found in %d locations:" % (libname,
len(paths)))
    for p in paths:
        print("- %r" % p)

check_library_duplicate(test_lib)
```

Recovering Data

Computer crashes, power outages, or simply forgetting to save can result in the loss or corruption of your work. You can use Blender's *Auto Save* feature to reduce the chance of losing files when such events occur.

There are options to save a backup of your files like *Auto Save* that saves your file automatically over time, and *Save on Quit*, which saves your blend-file automatically when you exit Blender. In addition to these functions being enabled by default, the *Save on Quit* functionality cannot be disabled.

Note

For your actions, there are options like *Undo*, *Redo* and an *Undo History*, used to roll back from mistakes under normal operation, or return back to a specific action. See Undo & Redo.

Recovering Save Versions

By default Blender keeps an additional backup when saving files. So saving renames the previously saved file with a `.blend1` extension instead of overwriting it.

This file can be used to revert to a previous state.

See Save Versions to configure the number of versions kept.

Recovering Auto Saves

Last Session

Reference

Menu:: *File* ▸ *Recover* ▸ *Last Session*

The *Recover Last Session* will open the `quit.blend` file that is saved into the Temporary Directory when you quit Blender under normal operation (see Blender Session). Note that files in your temporary directory may be deleted when you reboot your computer (depending on your system configuration).

Auto Save

Reference

Menu:: *File* ▸ *Recover* ▸ *Auto Save*

The *Recover Auto Save* allows you to open the *Auto Saved* file. You will have to navigate to your Temporary Directory. The *Auto Saved* files are named using a random number and have a blend extension.

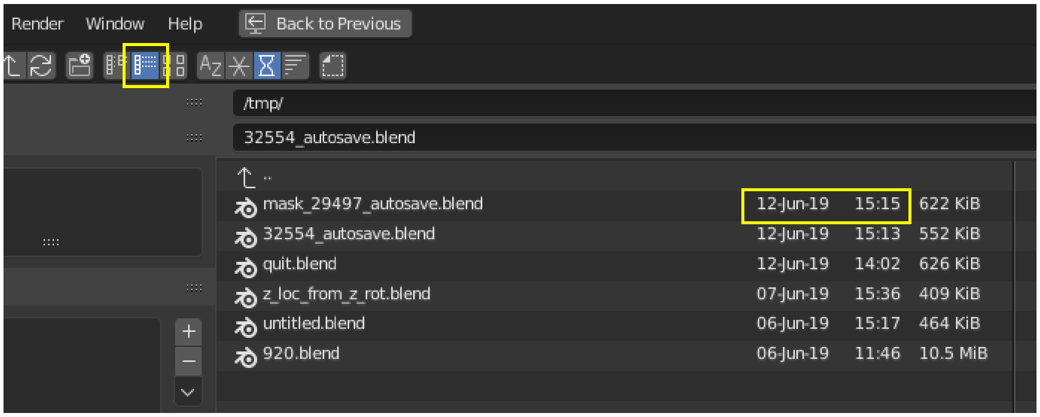
See Auto Save Preferences to configure auto-save.

Trusted Source

When enabled, Python scripts and drivers that may be included in the file will be run automatically. Enable this only if you created the file yourself, or you trust that the person who gave it to you did not include any malicious code with it. See Python Security to configure default trust options.

Tip

Enable the detailed list view when browsing auto-saved files to show which is the most recent.



/tmp/

32554_autosave.blend

..

mask_29497_autosave.blend	12-Jun-19	15:15	622 KiB
32554_autosave.blend	12-Jun-19	15:13	552 KiB
quit.blend	12-Jun-19	14:02	626 KiB
z_loc_from_z_rot.blend	07-Jun-19	15:36	409 KiB
untitled.blend	06-Jun-19	15:17	464 KiB
920.blend	06-Jun-19	11:46	10.5 MiB

File Browser displaying a vertical list.

Warning

When recovering an *Auto Saved* file, any changes made since the last *Auto Save* will be lost. Only one *Auto Saved* file exists for each `.blend` file, i.e. Blender does not keep older versions. Therefore, you will only be able to restore the most recent *Auto Save* file.

Glossary

This page lists definitions for terms used in Blender and this manual.

Action Safe

Area of the screen visible on most devices. Place content inside it to ensure it does not get cut off.

Active

When many items are selected, the last selected item will be the active one. Used in situations where the interface only shows options for one item at a time.

See also selection states.

Aliasing

Rendering artifacts in the form of jagged lines.

Alpha Channel

Additional channel in an image for transparency.

Straight Alpha

Method where RGBA channels are stored as (R, G, B, A) channels, with the RGB channels unaffected by the alpha channel. This is the alpha type used by paint programs such as Photoshop or Gimp, and used in common file formats like PNG, BMP or Targa. So, image textures or output for the web are usually straight alpha.

Premultiplied Alpha

Method where RGBA channels are stored as (R × A, G × A, B × A, A), with the alpha multiplied into the RGB channel.

This is the natural output of render engines, with the RGB channels representing the amount of light that comes toward the viewer, and alpha representing how much of the light from the background is blocked. The OpenEXR file format uses this alpha type. So, intermediate files for rendering and compositing are often stored as premultiplied alpha.

Conversion (Straight/Premultiplied) Alpha

Conversion between the two alpha types is not a simple operation and can involve data loss, as both alpha types can represent data that the other cannot, though it is often subtle.

Straight alpha can be considered to be an RGB color image with a separate alpha mask. In areas where this mask is fully transparent, there can still be colors in the RGB channels. On conversion to premultiplied alpha, this mask is *applied* and the colors in such areas become black and are lost.

Premultiplied alpha, on the other hand, can represent renders that are both emitting light and letting through light from the background. For example, a transparent fire render might be emitting light, but also letting through all light from objects behind it. On converting to straight alpha, this effect is lost.

Channel Packed

A separate image map is stored for each color and alpha channel. Channel packing is commonly used by game engines to save memory and to optimize memory access.

Ambient Light

The light that comes from the surrounding environment as a whole.

Ambient Occlusion

A ratio of how much Ambient Light a surface point would be likely to receive. If a surface point is under a foot or table, it will end up much darker than the top of someone's head or the tabletop.

Animation

Simulation of motion.

Anti-Aliasing

Is the technique of minimizing Aliasing, by e.g. rendering multiple samples per pixel.

Armature

An Object consisting of Bones. Used to Rig characters, props, etc.

Asset

Curated data-blocks that are meant for reuse, usually contained in an Asset Library. See also Asset Libraries.

Note that there are other meanings of the word “asset” – sometimes this is used more generically, and refers to any “useful thing”, like images, models, materials, and more.

Asset Catalog

Container for assets, similar to what a directory is for files. See also Asset Catalogs.

Asset Library

Directory on drive, registered in the list of asset libraries in the preferences. See also Asset Libraries and Current File Asset Library.

Asset Metadata

Asset-related information, such as its catalog, description, author, preview, and tags. See Asset Details Region.

Attribute

A generic term to describe data stored per-element in a geometry data-block.

Axis

A reference line which defines coordinates along one cardinal direction in n-dimensional space.

Axis Angle

Rotation method where X, Y, and Z correspond to the axis definition, while W corresponds to the angle around that axis, in radians.

Baking

The process of computing and storing the result of a potentially time-consuming calculation so as to avoid needing to calculate it again.

Bevel

The operation to chamfer or bevel edges of an object.

Bézier

A computer graphics technique for generating and representing curves.

Bit Depth

The exponent value (with base two) for how many colors can be represented within a single color channel. A higher bit depth will allow more possible colors, reducing banding, and increasing precision. Yet a higher bit depth will increase memory usage exponentially.

Blend Modes
Color Blend Modes

Methods for blending two colors together.

See also Blend Modes on Krita docs.

Blender Session
Session

The timespan of a Blender instance. The session begins with starting an instance of Blender and ends with closing it.

In some cases, loading a new file may be considered beginning a new session. If so, the documentation should mention that.

Bone

The building block of an Armature. Made up of a Head, Tail and Roll Angle which define a set of local axes and a point of rotation at the

Head. Also see Pose Bone.

Bone Collection

Collection of bones of an Armature, identified by its name. Bone collections can be used to organise bones and toggle their visibility. See Bone Collections.

Boolean

A type of logic dealing with binary true/false states.

See also Boolean Modifier.

Bounding Box

The box that encloses the shape of an object. The box is aligned with the local space of the object.

Bump Mapping

Technique for simulating slight variations in surface height using a grayscale “heightmap” texture.

BVH
Bounding Volume Hierarchy

A hierarchical structure of geometric objects.

See also Bounding Volume Hierarchy on Wikipedia.

Caustics

The optical phenomenon of light concentration focused by specular reflections or refracting objects. In example observable on light passing through a glass of water onto a table or the pattern at the bottom of a swimming pool.

In rendering this refers to diffuse reflected light paths after a glossy or refraction bounce.

See also Caustics on Wikipedia.

Child

An Object that is affected by its Parent.

Chroma
Chrominance

In general, a resulting image color decomposition, where its (*L* or *Y*) luminance channel is separated. There are two different contexts whereas this term is used:

Video Systems

Refers to the general color decomposition resulting in *Y* (Luminance) and *C* (Chrominance) channels, whereas the chrominance is represented by: U = (Blue minus Luminance) and V = (Red minus Luminance).

Matte Compositing

Refers to a point in the color gamut surrounded by a mixture of a determined spectrum of its RGB neighboring colors. This point is called *Chroma key* and this key (a chosen color) is used to create an *Alpha Mask*. The total amount of gamut space for this chrominance point is defined by users in a circular or square-shaped format.

Chromaticities

The coordinates of the Primaries on the CIE 1931 xy chromaticity diagram.

Clamp
Clamping

Limits a variable to a range. The values over or under the range are set to the constant values of the range's minimum or maximum.

Collection

A device for organizing objects. See also Collections.

Color Gamut

A gamut traditionally refers to the volume of color a particular color model/space can cover. In many instances, it is illustrated via a 2D model using CIE Yxy coordinates.

Color Model

A mechanism for representing colors as numbers.

RGB

An additive system where three primaries; red, green, and blue are combined to make other colors.

HSV

Three values often considered as more intuitive (human perception) than the RGB system. In this model, colors are represented as Hue, Saturation, and Value.

HSL

Similar to *HSV* except the colors are represented as Hue, Saturation, and Luminance.

YUV

Luminance-Chrominance standard used in broadcasting analog PAL (European) video.

YCbCr

Luminance-ChannelBlue-ChannelRed component video for digital broadcast use, whose standards have been updated for HDTV and commonly referred to as the HDMI format for component video.

Color Space

A coordinate system in which a vector represent a color value. This way the color space defines three things:

- The exact color of each of the Primaries
- The White Point
- A transfer function

The color spaces supported by Blender depend on the active OCIO config. The default supported color spaces are described in detail here: Default OpenColorIO Configuration

sRGB

A color space that uses the Rec .709 Primaries and a D65 white point, and 2.2 gamma correction value as the transfer function.

Concave Face

Face in which one vertex is inside a triangle formed by other vertices of the face.

See also Convex and concave polygons on Wikipedia.

Constraint

A way of controlling one Object with data from another.

Convex Face

Face where, if lines were drawn from each vertex to every other vertex, all lines would remain in the face. Opposite of a Concave Face.

Coplanar

Refers to any set of elements that are all aligned to the same 2D plane in 3D space.

Crease

Property of an Edge. Used to define the sharpness of edges in Subdivision Surface meshes.

Current File Asset Library

Asset library that is not a directory on drive, but only reflects the assets in the current blend-file. This library is available regardless of the location of the blend-file. See The Current File Asset Library.

Curve

A type of object defined in terms of a line interpolated between Control Vertices. Available types of curves include Bézier, NURBS and Poly.

Curve Segment

The part of a curve connecting two adjacent control points.

Cyclic

Often referring to an object being circular. This term is often associated with Curve.

Data User

An existing Blender object, which is using its own data, or linked data (data owned and controlled by another Blender object).

Dielectric Material

A material for real world objects that are electrical insulators such as plastics, wood, glass, ect. Essentially this summarizes any material that is solid and non metallic.

Diffuse Light

Even, directed light coming off a surface. For most things, diffuse light is the main lighting we see. Diffuse light comes from a specific direction or location and creates shading. Surfaces facing towards the light source will be brighter, while surfaces facing away from the light source will be darker.

Directional Light

The light that has a specific direction, but no location. It seems to come from an infinitely far away source, like the sun. Surfaces facing the light are illuminated more than surfaces facing away, but their location does not matter. A directional light illuminates all objects in the scene, no matter where they are.

Displacement Mapping

A method for distorting vertices based on an image or texture. Similar to Bump Mapping, but instead operates on the mesh's actual geometry.

This relies on the mesh having enough geometry to represent details in the image.

Display Referenced

Refers to an image whose Luminance channel is limited to a certain range of values (usually 0-1). The reason it is called display referenced is because a display cannot display an infinite range of values. So, the term Scene Referenced must go through a transfer function to be converted from one to the other.

DOF
Depth of Field

The distance in front of and behind the subject which appears to be in focus. For any given lens setting, there is only one distance at which a subject is precisely in focus, but focus falls off gradually on either side of that distance, so there is a region in which the blurring is tolerable. This region is greater behind the point of focus than it is in front, as the angle of the light rays change more rapidly; they approach being parallel with increasing distance.

Double Buffer

Technique for rendering and displaying content on the screen. Blender uses two buffers (images) to render the interface, the content of one buffer is displayed while rendering occurs on the other buffer. When rendering is complete, the buffers are switched.

Edge

Straight segment (line) that connects two Vertices, and can be part of a Face.

Edge Loop

Chain of Edges belonging to consecutive Quads. An edge loop ends at a pole or a boundary. Otherwise, it is cyclic.

Edge Ring

Path of all Edges along a Face Loop that share two faces belonging to that loop.

Elastic

Objects that are able to spontaneously return to their original shape after all outside forces are removed from the object.

Elasticity

The amount a material is elastic versus inelastic.

Empty

An Object without any Vertices, Edges or Faces.

Euler
Euler Rotation

Rotation method where rotations are applied to each of the X, Y, Z axes in a specific order.

Euler orders in Blender are most intuitive when read backwards: *XYZ Euler* is similar to rotating around *Local Z* using the *Rotate* tool in the 3D Viewport, followed by *Local Y* and then *Local X*.

F-Curve

A curve that holds the animation values of a specific property.

Face

Mesh element that defines a piece of surface. It consists of three or more Edges.

Face Loop

Chain of consecutive Quads. A face loop stops at a Triangle or N-gon (which do not belong to the loop), or at a boundary. Otherwise, it is cyclic.

Face Normal

The normalized vector perpendicular to the plane that a Face lies in. Each face has its own normal.

Fake User

A special Data User, a program construct that is used to mark an object (e.g. material) to be saved in a blend-file, even when no Real User is using the object. Objects that are not used by any Data User are not included in saved blend-files.

Field of View

The area in which objects are visible to the camera. Also see Focal Length.

Fireflies

Rendering artifacts encountered with path tracing resulting from improbable samples that contribute very high values to pixels.

FK

Forward Kinematics

The process of determining the movement of interconnected segments or bones of a body or model in the order from the parent bones to the child bones. Using forward kinematics on a hierarchically structured object, you can move the upper arm then the lower arm and hand go along with the movement. Without forward kinematics the lower arm and hand would disconnect from upper arm and would move independently in space.

See also Inverse Kinematics.

Focal Length

The distance required by a lens to focus collimated light. Defines the magnification power of a lens. Also see Field of View.

Frame Types

In video compression, a frame can be compressed by several different algorithms. These algorithms are known as *picture types* or *frame types* and there are three major types: **I**, **P**, and **B** frames.

I-frames

The least compressible but don't require other video frames to decode.

P-frames

Use data from previous frames to decompress and are more compressible than I-frames.

B-frames

Use both previous and forward frames for data reference to get the highest amount of compression.

Gamma

An operation used to adjust the brightness of an image.

See also Gamma correction on Wikipedia.

Geodesic

Relating to the shortest possible path between two points on a curved surface.

Geometric Center

The mean average of the positions of all vertices making up the object.

Gimbal

A pivoted support that allows the rotation of an object about a single axis.

See also Gimbal on Wikipedia.

Gimbal Lock

The limitation where axes of rotation can become aligned, losing the ability to rotate on an axis (typically associated with Euler Rotation).

- See also Gimbal lock on Wikipedia.

- See also Gimbal lock on Stack Exchange.

Global Illumination

A superset of Radiosity and ray tracing. The goal is to compute all possible light interactions in a given scene, and thus, obtain a truly photorealistic image. All combinations of diffuse and specular reflections and transmissions must be accounted for. Effects such as color bleeding and caustics must be included in a global illumination simulation.

Global Space

See World Space.

Glossy Map

See Roughness Map.

HDRI

High Dynamic Range Image

A set of techniques that allow a far greater dynamic range of exposures than normal digital imaging techniques. The intention is to accurately represent the wide range of intensity levels found in real scenes, ranging from direct sunlight to the deepest shadows.

See also HDRI on Wikipedia.

Head

A subcomponent of a Bone. The point of rotation for the bone has X, Y, and Z coordinates measured in the Local Space of the Armature object. Used in conjunction with the Tail to define the local Y axis of the bone in Pose Mode. The larger of the two ends when displayed as an Octahedron.

Hue

A shade of light out of the color spectrum.

IK

Inverse Kinematics

The process of determining the movement of interconnected segments or bones of a body or model in the order from the child bones to the parent bones. Using inverse kinematics on a hierarchically structured object, you can move the hand then the upper and lower arm will automatically follow that movement. Without inverse kinematics the hand would come off the model and would move independently in space.

See also Forward Kinematics.

Interpolation

The process of calculating new data between points of known value, like Keyframes.

IOR
Index of Refraction

A property of transparent materials. When a light ray travels through the same volume it follows a straight path. However, if it passes from one transparent volume to another, it bends. The angle by which the ray is bent can be determined by the IOR of the materials of both volumes.

Keyframe

A frame in an animated sequence drawn or otherwise constructed directly by the animator. In classical animation, when all frames were drawn by animators, the senior artist would draw these frames, leaving the “in between” frames to an apprentice. Now, the animator creates only the first and last frames of a simple sequence (keyframes); the computer fills in the gap.

Keyframing

Inserting Keyframes to build an animated sequence.

Lattice

A type of object consisting of a non-renderable three-dimensional grid of vertices.

See also Lattice Modifier.

Light Bounces

Refers to the reflection or transmission of a light ray upon interaction with a material. See also Light Paths.

Local Space

A 3D coordinate system that originates (for Objects) at the Object Origin. or (for Bones) at the Head of the Bone.

Compare to World Space.

Luminance

The intensity of light either in an image/model channel, or emitted from a surface per square unit in a given direction.

Manifold

Manifold meshes, also called ‘water-tight’ meshes, define a closed non-self-intersecting volume (see also Non-manifold). A manifold mesh is a mesh in which the structure of the connected faces in a closed volume will always point the normals (and their surfaces) to the outside or to the inside of the mesh without any overlaps. If you recalculate those normals, they will always point at a predictable direction (to the outside or to the inside of the volume). When working with non-closed volumes, a manifold mesh is a mesh in which the normals will always define two different and non-consecutive surfaces. A manifold mesh will always define an even number of non-overlapped surfaces.

MatCap

Stands for “material capture”, using an image to represent a complete material including lighting and reflections.

Matte

Mask

A grayscale image used to include or exclude parts of an image. A matte is applied as an Alpha Channel, or it is used as a mix factor when applying Color Blend Modes.

Mesh

Type of object consisting of Vertices, Edges and Faces.

Micropolygons

A polygon roughly the size of a pixel or smaller.

MIP
Mip-map
Mip-mapping

'MIP' is an acronym of the Latin phrase 'multum in parvo', meaning 'much in little'. Mip-maps are progressively lower resolution representations of an image, generally reduced by half squared interpolations using Anti-Aliasing. Mip-mapping is the process used to calculate lower resolutions of the same image, reducing memory usage to help speed visualization, but increasing memory usage for calculations and allocation. Mip-mapping is also a process used to create small anti-aliased samples of an image used for texturing. The mip-mapping calculations are made by CPUs, but modern graphic processors can be selected for this task and are way faster.

See the mip-map option present in the System Preferences.

MIS
Multiple Importance Sampling

A process of estimating the direction of light rays to improve sampling quality.

See Multiple Importance Sampling and also Importance sampling on Wikipedia.

Modifiers

A non-destructive operation that is applied on top of some sort of data.

Motion Blur

The phenomenon that occurs when we perceive a rapidly moving object. The object appears to be blurred because of our persistence of vision. Simulating motion blur makes computer animation appear more realistic.

Multisampling

Rendering multiple samples per pixel, for Anti-Aliasing.

N-gon

A Face that contains more than four Vertices.

NDOF
3D Mouse

A general term used to describe a 3D mouse, or any input devices which supports more degrees of freedom than a conventional 2D input device, see: Touchpad.

Non-manifold

Non-Manifold meshes essentially define geometry which cannot exist in the real world. This kind of geometry is not suitable for several types of operations, especially those where knowing the volume (inside/outside) of the object is important (refraction, fluids, Boolean operations, or 3D printing, to name a few). A non-manifold mesh is a mesh in which the structure of a non-overlapped surface (based on its connected faces) will not determine the inside or the outside of a volume based on its normals, defining a single surface for both sides, but ended with flipped normals. When working with non-closed volumes, a non-manifold mesh will always determine at least one discontinuity in the normal directions, either by an inversion of a connected loop, or by an odd number of surfaces. A non-manifold mesh will always define an odd number of surfaces.

There are several types of non-manifold geometry:

- Some borders and holes (edges with only a single connected face), as faces have no thickness.

- Edges and vertices not belonging to any face (wire).
- Edges connected to three or more faces (interior faces).
- Vertices belonging to faces that are not adjoining (e.g. two cones sharing the vertex at the apex).

See also: Select Non-Manifold tool.

Nonlinear Animation

Animation technique that allows the animator to edit motions as a whole, not just the individual keys. Nonlinear animation allows you to combine, mix, and blend different motions to create entirely new animations.

Normal

The normalized vector perpendicular to a surface.

Normals can be assigned to vertices, faces and modulated across a surface using Normal Mapping.

See also Normals on Wikipedia.

Normal Mapping

Is similar to Bump Mapping, but instead of the image being a grayscale heightmap, the colors define in which direction the normal should be shifted, the three color channels being mapped to the three directions X, Y and Z. This allows more detail and control over the effect.

NURBS
Non-uniform Rational Basis Spline

A computer graphics technique for generating and representing curves and surfaces.

Object

Container for a type (mesh, curve, surface, metaball, text, armature, lattice, empty, camera, light) and basic 3D transform data (Object Origin).

Object Center
Object Origin

A reference point used to position, rotate, and scale an Object and to define its Local Space coordinates.

Octahedron

An eight-sided figure commonly used to depict the Bones of an Armature.

OpenGL

The graphics system used by Blender (and many other graphics applications) for rendering 3D graphics, often taking advantage of hardware acceleration.

See also OpenGL on Wikipedia.

Operator

An executable action that is completed the moment they're initiated. See Operators as described in the user interface section.

Overscan

The term used to describe the situation. when not all of a televised image is present on a viewing screen.

See also Overscan on Wikipedia.

Panel

A user interface element that contains buttons. Panels are collapsible to hide there contents and can often be rearranged. See Panels as described in the user interface section.

Parent

An Object that affects its Child objects.

Parenting

Creating a Parent-Child relationship between two objects.

Particle System

Technique that simulates certain kinds of fuzzy phenomena, which are otherwise very hard to reproduce with conventional rendering techniques. Common examples include fire, explosions, smoke, sparks, falling leaves, clouds, fog, snow, dust, meteor tails, stars, and galaxies, or abstract visual effects like glowing trails, magic spells. Also used for things like fur, grass or hair.

Phong

Local illumination model that can produce a certain degree of realism in three-dimensional objects by combining three elements: diffuse, specular and ambient for each considered point on a surface. It has several assumptions – all lights are points, only surface geometry is considered, only local modeling of diffuse and specular, specular color is the same as light color, ambient is a global constant.

Pivot Point

The pivot point is the point in space around which all rotation, scaling and mirror transformations are centered.

See also the Pivot Point docs.

Pixel

The smallest unit of information in a 2D raster image, representing a single color made up of red, green, and blue channels. If the image has an Alpha Channel, the pixel will contain a corresponding fourth channel.

Point Cloud

A list of points in 3D space.

Pole

Vertex where three, five, or more edges meet. A vertex connected to one, two, or four edges is not a pole.

Pose Bone

Pose-specific properties of a Bone, such as its location / rotation / scale relative to the Armature's rest pose. Its properties are stored on the Object, and thus can be different for each user of the Armature. The Pose Bone also stores constraints.

Pose Mode

Used for Posing, Keyframing, Weight Painting, Constraining and Parenting the Bones of an Armature.

Posing

Moving, Rotating and Scaling the Pose Bones of an Armature to achieve an aesthetically pleasing pose for a character.

Premultiplied Alpha

See Alpha Channel.

Primaries

In color theory, primaries (often known as primary colors) are the abstract lights, using an absolute model, that make up a Color Space.

Primitive

A basic object that can be used as a basis for modeling more complicated objects.

Procedural Texture

Computer generated (generic) textures that can be configured via different parameters.

Projection

In computer graphics, there are two common camera projections used.

Perspective

A *perspective* view is geometrically constructed by taking a scene in 3D and placing an observer at point O. The 2D perspective scene is built by placing a plane (e.g. a sheet of paper) where the 2D scene is to be rendered in front of point O, perpendicular to the viewing direction. For each point *P* in the 3D scene a *PO* line is drawn, passing by *O* and *P*. The intersection point *S* between this *PO* line and the plane is the perspective projection of that point. By projecting all points *P* of the scene you get a perspective view.

Orthographic

In an *orthographic* projection, you have a viewing direction but not a viewing point *O*. The line is then drawn through point *P* so that it is parallel to the viewing direction. The intersection *S* between the line and the plane is the orthographic projection of the point *P*. By projecting all points *P* of the scene you get the orthographic view.

Proxy

For video editing, a proxy is a smaller version of the original file, typically using an optimized video codec and lower resolution version (faster to load) that stands in for the main image or video.

When proxies are built, editing functions like scrubbing and scrolling and compositing is much faster but gives lower resolution and slightly imprecise result.

Quad
Quadrilateral
Quadrangle

Face that contains exactly four Vertices.

Quaternion
Quaternion Rotation

Rotation method where rotations are defined by four values (X, Y, Z, and W). X, Y, and Z also define an Axis, and W an angle, but it is quite different from Axis Angle.

Quaternion values can be interpreted geometrically as defining a point on a unit sphere in 4D space. Moving along any *great circle* of the sphere represents rotating around a fixed axis, with one full circle matching two full rotations.

Radiosity

A global lighting method that calculates patterns of light and shadow for rendering graphics images from three-dimensional models. One of the many different tools which can simulate diffuse lighting in Blender.

See also Radiosity (computer graphics) on Wikipedia.

Random Seed
Seed

Blender uses pseudo random number generators, which produce numbers that appear to be random, but given the same initial condition, they will always produce the exact same sequence of numbers.

This is a critical feature to get reproducible and/or stable effects (otherwise e.g. your hair simulation would change every time you re-run it, without any way to control the outcome).

The **seed** is a number that represents the initial condition of a random generator, if you change its seed, it will produce a new sequence of pseudo-random numbers.

See also Random seed on Wikipedia.

Ray Tracing

Rendering technique that works by tracing the path taken by a ray of light through the scene, and calculating reflection, refraction, or absorption of the ray whenever it intersects an object in the world. More accurate than Scanline, but much slower.

Real User

A Blender object, which is a Data User. Opposite of Fake User, which is only a program construct.

Refraction

The change in direction of a wave due to a change in velocity. It happens when waves travel from a medium with a given Index of Refraction to a medium with another. At the boundary between the media, the wave changes direction; its wavelength increases or decreases but frequency remains constant.

Render

The process of computationally generating a 2D image from 3D geometry.

Resource

External files such as images, sounds, fonts and volumes files that can be packed into a blend-file.

RGB

A color model based on the traditional primary colors, Red/Green/Blue. RGB colors are also directly broadcasted to most computer monitors.

Rig

A system of relationships that determine how something moves. The act of building of such a system.

Roll
Roll Angle

The orientation of the local X and Z axes of a Bone. Has no effect on the local Y axis as local Y is determined by the location of the Head and Tail.

Rolling Shutter

In real CMOS cameras the sensor is read out with scanlines and hence different scanlines are sampled at a different moment in time. This, for example, make vertical straight lines being curved when doing a horizontal camera pan. See also Rolling Shutter on Wikipedia.

Roughness Map

A grayscale texture that defines how rough or smooth the surface of a material is. This may also be known as a Glossy Map.

Saturation

Also known as colorfulness, saturation is the quantity of hue in the color (from desaturated – a shade of gray – to saturated – brighter colors).

Scanline

Rendering technique. Much faster than Ray Tracing, but allows fewer effects, such as reflections, refractions, motion blur and focal blur.

Scene Referenced

An image whose Luminance channel is not limited.

See also Display Referenced.

Shading

Process of altering the color of an object/surface in the 3D scene, based on its angle to lights and its distance from lights to create a photorealistic effect.

Smoothing

Defines how Faces are shaded. Faces can be either solid (faces are rendered flat) or smooth (faces are smoothed by interpolating the normal on every point of the face).

Specular Light

A light which is reflected precisely, like a mirror. Also used to refer to highlights on reflective objects.

SSS
Subsurface Scattering

Mechanism of light transport in which light penetrates the surface of a translucent object, is scattered by interacting with the material, and exits the surface at a different point. All non-metallic materials are translucent to some degree. In particular, materials such as marble, skin, and milk are extremely difficult to simulate realistically without taking subsurface scattering into account.

Straight Alpha

See Alpha Channel.

Subdiv
Subdivision Surface

A method of creating smooth higher poly surfaces which can take a low polygon mesh as input.

See also Catmull-Clark subdivision surface on Wikipedia.

Subdividing

Technique for adding more geometry to a mesh. It creates new vertices on subdivided edges, new edges between subdivisions and new faces based on new edges. If new edges cross a new vertex is created at their crossing point.

Swing
Swing and Twist

Refers to decomposition of an arbitrary rotation into a sequence of two single axis rotations: a *swing* rotation that aims a chosen axis in its final direction using the shortest possible rotation path, followed by a *twist* rotation around that axis.

This decomposition is available through Driver Variables and inputs of the Transformation constraint. The Damped Track constraint produces a pure *swing* rotation.

In the Quaternion representation the *swing* rotation always has 0 as the X/Y/Z component corresponding to the selected axis, while *twist* always has 0 as the other two components.

Tail

A subcomponent of a Bone. Has X, Y and Z coordinates measured in the Local Space of the armature object. Used in conjunction with the Head to define the local Y axis of a bone in Pose Mode. The smaller of the two ends when displayed as an Octahedron.

Tangent

A line that intersects a surface at exactly one point, a tangent is perpendicular to a Normal.

Tessellation

The tiling of a plane using one or more geometric shapes usually resulting in Micropolygons.

Texture

Specifies visual patterns on surfaces and simulates physical surface structure.

Texture Space

The bounding box to use when using *Generated* mapping to add a Texture to an image.

Timecode

A coded signal on videotape or film giving information about the frame number and time the frame was recorded. Timecodes are used to sync media between different recording devices, including both audio and video.

Title Safe

Area of the screen visible on all devices. Place text and graphics inside this area to make sure they do not get cut off.

Topology

The arrangement of *Vertices*, *Edges*, and *Faces* which define the shape of a mesh. See Vertex, Edge, and Face.

Transform
Transformation

The combination of location, rotation, and scale. Can be expressed in World Space or Local Space.

Transformation Matrix

A matrix that is used to represent the Transformation of an item.

Triangle

Face with exactly three Vertices.

UV Map

Defines a relation between the surface of a mesh and a 2D texture. In detail, each face of the mesh is mapped to a corresponding face on the texture. It is possible and often common practice to map several faces of the mesh to the same or overlapping areas of the texture.

Value

The brightness of the color (dark to light).

Vertex
Vertices

A point in 3D space containing a location. Vertices are the terminating points of Edges.

Vertex Group

Collection of Vertices. Vertex groups are useful for limiting operations to specific areas of a mesh.

Voxel

A cubic 3D equivalent to the square 2D pixel. The name is a combination of the terms “Volumetric” and “Pixel”. Used to store smoke and fire data from physics simulations.

Walk Cycle

In animation, a walk cycle is a character that has just the walking function animated. Later on in the animation process, the character is placed in an environment and the rest of the functions are animated.

Weight Painting

Assigning Vertices to a Vertex Group with a weight of 0.0 - 1.0.

White Point

A reference value for white light when all primaries of a color model are combined evenly.

A white point is defined by a set of CIE illuminates which correspond to a color temperature. For example, D65 corresponds to 6500 K light and D70 corresponding to 7000 K.

World Space

A 3D coordinate system that originates at a point at the origin of the world. Compare to Local Space.

Z-buffer

Raster-based storage of the distance measurement between the camera and the surface points. Surface points which are in front of the camera have a positive Z value and points behind have negative values. The Z-depth map can be visualized as a grayscale image.

Index

Symbols | A | B | C | D | E | F | G | H | I | K | L | M | N | O | P | Q | R | S | T | U | V | W | Z

Symbols

- **3D Mouse**

A

- **Action Safe**
- **Active**
- Add-ons, [1]
- Add-ons Extensions
- **Aliasing**
- **Alpha Channel**
- **Ambient Light**
- **Ambient Occlusion**
- **Animation**
- **Anti-Aliasing**
- **Armature**
- **Asset**
- **Asset Catalog**
- **Asset Library**
- **Asset Metadata**
- **Attribute**
- **Axis**
- **Axis Angle**

B

- **Baking**
- **Bevel**
- **Bézier**
- **Bit Depth**
- **Blend Modes**
- **Blender Session**
- **Bone**
- **Bone Collection**
- **Boolean**
- **Bounding Box**
- **Bounding Volume Hierarchy**
- bssrdf()
  - built-in function
- built-in function
  - bssrdf()
- **Bump Mapping**
- **BVH**

C

- **Caustics**
- **Child**
- **Chroma**
- **Chromaticities**
- **Chrominance**
- **Clamp**
- **Clamping**
- **Collection**
- **Color Blend Modes**
- **Color Gamut**
- **Color Model**
- **Color Space**
- Compositor Nodes
  - Alpha Convert
  - Alpha Over
  - Anti-Aliasing
  - Bilateral Blur
  - Blur
  - Bokeh Blur
  - Bokeh Image
  - Box Mask
  - Brightness/Contrast
  - Channel Key
  - Chroma Key
  - Color Balance
  - Color Correction
  - Color Key
  - Color Ramp
  - Color Space
  - Color Spill
  - Combine Color
  - Combine XYZ
  - Composite
  - Corner Pin
- **Concave Face**
- **Constraint**
  - Bone Constraints
  - Object Constraints
  - Rigid Body Constraints
- **Convex Face**
- **Coplanar**
- **Crease**
- **Current File Asset Library**
- **Curve**
- **Curve Segment**
- **Cyclic**

Crop
Cryptomatte
Cryptomatte (Legacy)
Defocus
Denoise
Despeckle
Difference Key
Dilate/Erode
Directional Blur
Displace
Distance Key
Double Edge Mask
Ellipse Mask
Exposure
File Output
Filter
Flip
Gamma
Glare
Hue Correct
Hue/Saturation/Value
ID Mask
Image
Inpaint
Invert Color
Keying
Keying Screen
Kuwahara
Lens Distortion
Levels
Luminance Key
Map Range
Map UV
Map Value
Mask
Math
Mix

- Movie Clip
- Movie Distortion
- Normal
- Normalize
- Pixelate
- Plane Track Deform
- Posterize
- Render Layers
- RGB
- RGB Curves
- RGB TO BW
- Rotate
- Scale
- Scene Time
- Separate Color
- Separate XYZ
- Set Alpha
- Split
- Stabilize 2D
- Sun Beams
- Switch
- Switch View
- Texture
- Time Curve
- Tone Map
- Track Position
- Transform
- Translate
- Vector Blur
- Vector Curves
- Viewer
- Z Combine

D

Data User

Directional Light

- **Depth of Field**
- **Dielectric Material**
- **Diffuse Light**
- **Displacement Mapping**
- **Display Referenced**
- **DOF**
- **Double Buffer**

E

- **Edge**
- **Edge Loop**
- **Edge Ring**
- Editors
  - 3D Viewport
  - Asset Browser
  - Compositor
  - Dope Sheet
  - Drivers Editor
  - File Browser
  - Geometry Node Editor
  - Graph Editor
  - Image Editor
  - Info Editor
  - Movie Clip Editor
  - NLA Editor
  - Outliner
  - Preferences
  - Properties
  - Python Console
  - Shader Editor
  - Spreadsheet
  - Text Editor
  - Texture Node Editor
  - Timeline
  - UV Editor
  - Video Sequencer
- **Elastic**
- **Elasticity**
- **Empty**
- **Euler**
- **Euler Rotation**
- Extensions, [1]

F

- **F-Curve**
- F-Curve Modifiers
  - Built-in Function Modifier
  - Cycles Modifier
  - Envelope Modifier
  - Generator Modifier
  - Limits Modifier
  - Noise Modifier
  - Stepped Interpolation Modifier
- **Face**
- **Face Loop**
- **Face Normal**
- **Fake User**
- **Field of View**
- **Fireflies**
- **FK**
- **Focal Length**
- Force Fields
  - Boid
  - Charge
  - Curve Guide
  - Drag
  - Fluid Force
  - Force
  - Harmonic
  - Lennard-Jones
  - Magnetic
  - Texture
  - Turbulence
  - Vortex
  - Wind
- **Forward Kinematics**
- **Frame Types**

G

- **Gamma**
- **Geodesic**
- **Geometric Center**
- Geometry Nodes
  - 3D Cursor
  - Accumulate Field
  - Active Camera
  - Active Element
  - Align Euler to Vector
  - Align Rotation to Vector
  - Arc
  - Attach Hair Curves to Surface
  - Attribute Reference
  - Attribute Statistic
  - Axis Angle to Rotation
- **Gimbal**
- **Gimbal Lock**
- **Global Illumination**
- **Global Space**
- **Glossy Map**
- Grease Pencil Modifiers
  - Armature Modifier
  - Array Modifier
  - Build Modifier
  - Dot Dash Modifier
  - Envelope Modifier
  - Hook Modifier
  - Hue/Saturation Modifier
  - Lattice Modifier
  - Length Modifier
  - Line Art Modifier

  - Axis to Rotation
  - Bake
  - Bézier Segment
  - Blackbody
  - Blend Hair Curves
  - Blur Attribute
  - Boolean
  - Boolean Math
  - Bounding Box
  - Braid Hair Curves
  - Brick Texture
  - Capture Attribute
  - Checker Texture
  - Clamp
  - Clump Hair Curves
  - Collection Info
  - Color
  - Color Ramp
  - Combine Color, [1]
  - Combine Matrix
  - Combine Transform
  - Combine XYZ
  - Compare
  - Cone
  - Convex Hull
  - Corners of Edge
  - Corners of Face
  - Corners of Vertex
  - Create Guide Index Map
  - Cube
  - Curl Hair Curves
  - Curve Circle
  - Curve Handle Position
  - Curve Info
  - Curve Length
  - Curve Line
  - Curve of Point
  - Mirror Modifier
  - Multiple Strokes
  - Noise Modifier
  - Offset Modifier
  - Opacity Modifier
  - Outline Modifier
  - Shrinkwrap Modifier
  - Simplify Modifier
  - Smooth Modifier
  - Subdivide Modifier
  - Texture Mapping Modifier
  - Thickness Modifier
  - Time Offset Modifier
  - Tint Modifier
  - Vertex Weight Angle Modifier
  - Vertex Weight Proximity Modifier
- Grease Pencil Visual Effects
  - Blur Visual Effect
  - Colorize Visual Effect
  - Flip Visual Effect
  - Glow Visual Effect
  - Pixelate Visual Effect
  - Rim Visual Effect
  - Shadow Visual Effect
  - Swirl Visual Effect
  - Wave Distortion Visual Effect

Curve Root
Curve Segment
Curve Spiral
Curve Tangent
Curve Tilt
Curve Tip
Curve to Mesh
Curve to Points
Curves to Grease Pencil
Cylinder
Deform Curves on Surface
Delete Geometry
Dial Gizmo
Displace Hair Curves
Distribute Points in Volume
Distribute Points on Faces
Domain Size
Dual Mesh
Duplicate Elements
Duplicate Hair Curves
Edge Angle
Edge Neighbors
Edge Paths to Curves
Edge Paths to Selection
Edge Split
Edge Vertices
Edges of Corner
Edges of Vertex
Edges to Face Groups
Endpoint Selection
Euler to Rotation
Evaluate at Index
Evaluate on Domain
Extrude Mesh
Face Area
Face Group Boundaries

Face Neighbors
Face of Corner
Face Set
Fields
Fill Curve
Fillet Curve
Flip Faces
Float Curve
Float To Integer
For Each Element
Frizz Hair Curves
Gabor Texture
Generate Hair Curves
Geometry Proximity
Geometry to Instance
Gizmos
Gradient Texture
Grease Pencil to Curves
Grid
Hair
Hair Attachment Info
Hair Curves Noise
Handle Type Selection
Hash Value
Ico Sphere
ID
Image
Image Info
Image Texture
Index
Index of Nearest
Index Switch
Inspection
Instance on Points
Instance Rotation
Instance Scale
Instance Transform

Instances
Instances to Points
Integer
Integer Math
Interpolate Curves
Interpolate Hair Curves
Invert Matrix
Invert Rotation
Is Edge Smooth
Is Face Planar
Is Face Smooth
Is Spline Cyclic
Is Viewport
Join Geometry
Join Strings
Linear Gizmo
Magic Texture
Map Range
Material
Material Index
Material Selection
Math
Matrix Determinant
Menu Switch
Merge by Distance
Merge Layers
Mesh Boolean
Mesh Circle
Mesh Island
Mesh Line
Mesh to Curve
Mesh to Points
Mesh to Volume
Mix
Mix Color
Mix Vector
Mouse Position

Multiply Matrices
Musgrave Texture
Named Attribute
Noise Texture
Normal
Object Info
Offset Corner in Face
Offset Point in Curve
Pack UV Islands
Points
Points of Curve
Points to Curves
Points to Vertices
Points to Volume
Position
Project Point
Quadratic Bézier
Quadrilateral
Quaternion to Rotation
Radius
Random Value
Raycast
Realize Instances
Redistribute Curve Points
Remove Named Attribute
Repeat
Replace Material
Replace String
Resample Curve
Restore Curve Segment
Length
Reverse Curve
RGB Curves
Roll Hair Curves
Rotate Euler
Rotate Hair Curves
Rotate Instances

Rotate Rotation
Rotate Vector
Rotation
Rotation to Euler
Rotation to Quaternion
Sample Curve
Sample Index
Sample Nearest
Sample Nearest Surface
Sample UV Surface
Scale Elements
Scale Instances
Scene Time
Selection
Self Object
Separate Color, [1]
Separate Components
Separate Geometry
Separate Matrix
Separate Transform
Separate XYZ
Set Curve Normal
Set Curve Radius
Set Curve Tilt
Set Face Set
Set Geometry Name
Set Hair Curve Profile
Set Handle Positions
Set Handle Type
Set ID
Set Instance Transform
Set Material
Set Material Index
Set Point Radius
Set Position
Set Selection
Set Shade Smooth

Set Spline Cyclic
Set Spline Resolution
Set Spline Type
Shortest Edge Paths
Shrinkwrap Hair Curves
Simulation
Slice String
Smooth Hair Curves
Sort Elements
Special Characters
Spline Length
Spline Parameter
Spline Resolution
Split To Instances
Star
Store Named Attribute
Straighten Hair Curves
String
String Length
String to Curves
Subdivide Curve
Subdivide Mesh
Subdivision Surface
Switch
Tools
Transform Direction
Transform Geometry
Transform Gizmo
Transform Point
Translate Instances
Transpose Matrix
Triangulate
Trim Curve
Trim Hair Curves
UV Sphere
UV Unwrap
Value

- Value to String
- Vector
- Vector Curves
- Vector Math
- Vector Rotate
- Vertex Neighbors
- Vertex of Corner
- Viewer
- Viewport Transform
- Volume Cube
- Volume to Mesh
- Voronoi Texture
- Warning
- Wave Texture
- White Noise Texture

H

- **HDRI**
- **Head**
- **High Dynamic Range Image**
- **Hue**

I

- **IK**
- **Index of Refraction**
- **Interpolation**
- **Inverse Kinematics**
- **IOR**

K

- **Keyframe**
- **Keyframing**

L

- **Lattice**
- **Light Bounces**

Licenses
Local Space
Luminance

M

Manifold
Mask
MatCap
Matte
Mesh
Micropolygons
MIP
Mip-map
Mip-mapping
MIS
Modeling
Modeling Modifiers
    Armature Modifier
    Array Modifier
    Bevel Modifier
    Boolean Modifier
    Build Modifier
    Cast Modifier
    Cloth Modifier
    Collision Modifier
    Curve Modifier
    Data Transfer Modifier
    Decimate Modifier
    Displace Modifier
    Dynamic Paint Modifier
    Edge Split Modifier
    Explode Modifier
    Fluid Modifier
    Geometry Nodes Modifier
    Hook Modifier
    Laplacian Deform Modifier
Modeling Transform
Modifiers
    F-Curve Modifiers
    Grease Pencil Modifiers
    Modeling Modifiers
    Video Sequencer Modifiers
Motion Blur
Multiple Importance Sampling
Multisampling

Lattice Modifier
Mask Modifier
Mesh Cache Modifier
Mesh Deform Modifier
Mesh Sequence Cache Modifier
Mesh to Volume
Mirror Modifier
Multiresolution Modifier
Normal Edit Modifier
Ocean Modifier
Particle Instance Modifier
Particle System Modifier
Remesh Modifier
Screw Modifier
Shrinkwrap Modifier
Simple Deform Modifier
Skin Modifier
Smooth By Angle Modifier
Smooth Corrective Modifier
Smooth Laplacian Modifier
Smooth Modifier
Soft Body Modifier
Solidify Modifier
Subdivision Surface Modifier
Surface Deform Modifier
Triangulate Modifier
UV Project Modifier
UV Warp Modifier
Vertex Weight Edit Modifier
Vertex Weight Mix Modifier
Vertex Weight Proximity Modifier

    - Volume Displace
    - Volume to Mesh
    - Warp Modifier
    - Wave Modifier
    - Weighted Normal Modifier
    - Weld Modifier
    - Wireframe Modifier

N

- **N-gon**
- **NDOF**
- Nodes
    - Compositing Nodes
    - Geometry Nodes
    - Shader Nodes
    - Texture Nodes
- **Non-manifold**
- **Non-uniform Rational Basis Spline**
- **Nonlinear Animation**
- **Normal**
- **Normal Mapping**
- **NURBS**

O

- **Object**
- **Object Center**
- Object Constraints
    - Action Constraint
    - Armature Constraint
    - Camera Solver Constraint
    - Child Of Constraint
    - Clamp To Constraint
    - Copy Location Constraint
    - Copy Rotation Constraint
    - Copy Scale Constraint
    - Copy Transforms Constraint
    - Damped Track Constraint
    - Floor Constraint
- **Object Origin**
- **Octahedron**
- **OpenGL**
- **Operator**
- **Overscan**

- Follow Path Constraint
- Follow Track Constraint
- Inverse Kinematics Constraint
- Limit Distance Constraint
- Limit Location Constraint
- Limit Rotation Constraint
- Limit Scale Constraint
- Locked Track Constraint
- Maintain Volume Constraint
- Object Solver Constraint
- Pivot Constraint
- Shrinkwrap Constraint
- Spline IK Constraint
- Stretch To Constraint
- Track To Constraint
- Transform Cache Constraint
- Transformation Constraint

P

- **Panel**
- **Parent**
- **Parenting**
- **Particle System**
- **Phong**
- **Pivot Point**
- **Pixel**
- **Point Cloud**
- **Pole**
- **Pose Bone**
- **Pose Mode**
- **Posing**
- **Premultiplied Alpha**
- **Primaries**
- **Primitive**
- **Procedural Texture**
- **Projection**
- **Proxy**

Q

Quad
Quadrangle
Quadrilateral
Quaternion
Quaternion Rotation

R

Radiosity
Random Seed
Ray Tracing
Real User
Refraction
Render
Resource
RGB
Rig
Rigid Body Constraints
    Fixed Constraint
    Generic Constraint
    Generic Spring Constraint
    Hinge Constraint
    Motor Constraint
    Piston Constraint
    Point Constraint
    Slider Constraint
Roll
Roll Angle
Rolling Shutter
Roughness Map

S

Saturation
Scanline
Scene Referenced
Seed
Session
Shader Nodes
    Combine XYZ
    Separate XYZ
Smoothing
Specular Light
SSS
Straight Alpha
Subdiv
Subdividing
Subdivision Surface
Subsurface Scattering

Shading
Swing
Swing and Twist

T

Tags
Tail
Tangent
Tessellation
Texture
Texture Space
Timecode
Title Safe
Topology
Transform
    Modeling Transform
Transformation
Transformation Matrix
Triangle

U

User Interface
UV Map

V

Value
Vertex
Vertex Group
Vertices
Video Sequencer Modifiers
    Brightness/Contrast Modifier
    Color Balance Modifier
    Curves Modifier
    Hue Correct Modifier
    Mask Modifier
    Sound Equalizer Modifier
    Tone Map Modifier
    White Balance Modifier
Voxel

W

- Walk Cycle
- Weight Painting
- White Point
- World Space

Z

- Z-buffer

Contribute Documentation

The Blender Manual is a community driven effort to which anyone can contribute. Whether you like to fix a tiny spelling mistake or rewrite an entire chapter, your help with the Blender manual is most welcome!

If you find an error in the documentation, please report the problem.

Get involved in discussions through any of the project Contacts.

Getting Started

The following guides lead you through the process.

- Installing Dependencies
- Building the Manual
- Editing the Manual
- Pull Requests

Where to help

- Todo List

Guidelines

- Writing Style Guide
- Markup Style Guide
- Commit Guidelines
- Templates
- Maintenance

Translations

- Contribute
- Style Guide

Contacts

Project Page
: An overview of the documentation project.

Documentation Forum
: A forum based discussions on writing and translating documentation. This includes the user manual, Wiki, release notes, and code docs.

Chat
: `#docs` channel for informal discussions in real-time.

Project Workboard
: Manage tasks such as bugs, todo lists, and future plans.

Installing Dependencies

This section documents how to install the software used to generate the manual. The installation is different for each operating system, instructions have been written for:

- Linux
- macOS
- Windows

Installation on Linux

This guide covers the following topics:

1. Installing Dependencies
2. Downloading the Repository
3. Setting up the Build Environment

Installing Dependencies

Below are listed the installation commands for popular Linux distributions.

For the appropriate system, run the command in a terminal:

Debian/Ubuntu:

```
sudo apt-get install python3 python3-pip git git-lfs
git lfs install --skip-repo
```

Redhat/Fedora:

```
sudo dnf install python python-pip git git-lfs
git lfs install --skip-repo
```

Arch Linux:

```
sudo pacman -S python python-pip git git-lfs
git lfs install --skip-repo
```

Downloading the Repository

Simply check out the Blender Manual’s repository using:

```
cd ~
git clone https://projects.blender.org/blender/blender-manual.git
```

The repository will now be downloaded which may take a few minutes depending on your internet connection.

Setting up the Build Environment

Tip

It is recommended to setup and activate a virtual Python environment where dependencies will be installed:

```
python3 -m venv .venv
source .venv/bin/activate
```

Repeat the `source .venv/bin/activate` command to re-activate the virtual environment, whenever you open a new terminal to build the documentation.

This step may be required on some distributions that enforce PEP 668.

- Open a Terminal window.
- Enter the `blender-manual` folder which was just added by `git clone`:

  ```
  cd ~/blender-manual
  ```

- Inside that folder is a file called `requirements.txt` which contains a list of all the dependencies we need. To install these dependencies, we can use the `pip3` command:

  ```
  pip3 install -r requirements.txt
  ```

Note

Every now and then you may want to make sure your dependencies are up to date using:

```
pip3 install -r requirements.txt --upgrade
```

Installation on macOS

This guide covers the following topics:

1. Installing Dependencies
2. Downloading the Repository
3. Setting up the Build Environment

Note

This guide relies heavily on command-line tools. It assumes you are the least familiar with the macOS Terminal application.

Installing Dependencies

Install those packages or make sure you have them in your system.

- PIP
- Git
- Git LFS

When using Homebrew, run the following commands in the terminal:

```
python3 -m ensurepip
brew install git git-lfs
git lfs install
```

Downloading the Repository

Simply check out the Blender Manual's repository using:

```
cd ~
git clone https://projects.blender.org/blender/blender-manual.git
```

The repository will now be downloaded which may take a few minutes depending on your internet connection.

Setting up the Build Environment

Tip

It is recommended to setup and activate a virtual Python environment where dependencies will be installed:

```
python3 -m venv .venv
source .venv/bin/activate
```

Repeat the `source .venv/bin/activate` command to re-activate the virtual environment, whenever you open a new terminal to build the documentation.

- Open a Terminal window.
- Enter the `blender-manual` folder which was just added by `git clone`:

  ```
  cd ~/blender-manual
  ```

- Inside that folder is a file called `requirements.txt` which contains a list of all the dependencies we need. To install these dependencies, we can use the `pip` command:

  ```
  pip install -r requirements.txt
  ```

Note

Every now and then you may want to make sure your dependencies are up to date using:

```
pip install -r requirements.txt --upgrade
```

Installation on Windows

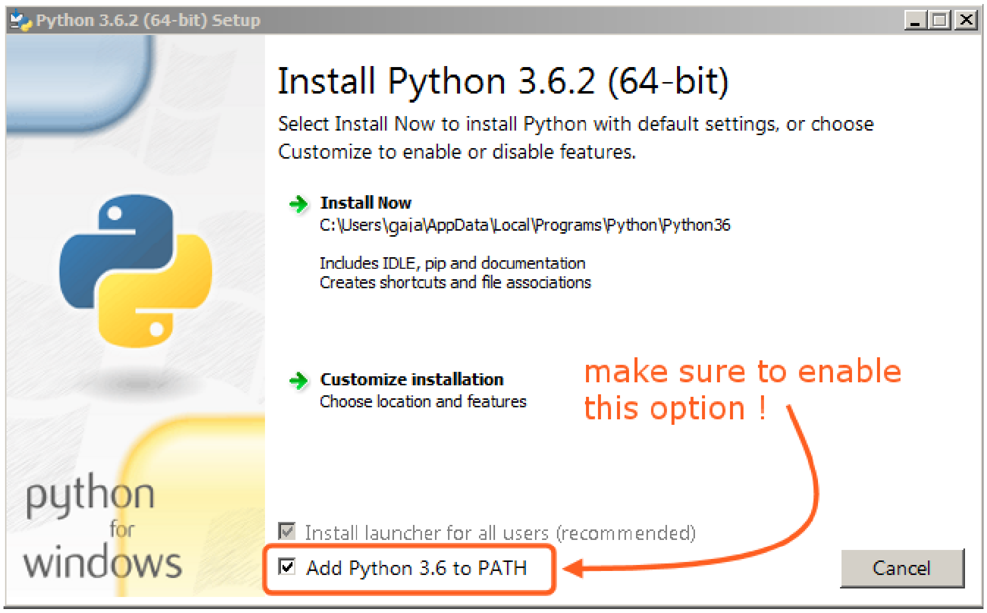
This guide covers the following topics:

1. Installing Python (used to “convert” the source files to HTML)
2. Installing Git and Downloading the Repository
3. Setting up the Build Environment

Installing Python

1. Download the Python installation package for Windows. In this guide version 3.9.x is used.

2. Install Python with the installation wizard. Please make sure that you enable the “Add Python to PATH” option:



Python 3.6.2 (64-bit) Setup

Install Python 3.6.2 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\gaia\AppData\Local\Programs\Python\Python36

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

make sure to enable this option !

☑ Install launcher for all users (recommended)
☑ Add Python 3.6 to PATH

The option must be enabled so you can build the manual with the make script.

All other settings can remain as set by default.

Installing Git and Downloading the Repository

In this guide, we will use the official Git client, though any Git client will do.

1. Download and install Git for Windows.
2. Simply check out the Blender Manual's repository using:

   ```
   cd ~
   git lfs install
   git clone https://projects.blender.org/blender/blender-
   manual.git
   ```

3. The repository will now be downloaded which may take a few minutes depending on your internet connection.

Note

This process can be completed using a graphical Git client, in that case you will just use the repository address in the URL field provided by your client:

```
https://projects.blender.org/blender/blender-manual.git
```

Setting up the Build Environment

Tip

It is recommended to setup and activate a virtual Python environment where dependencies will be installed:

```
python3 -m venv .venv
.venv/Scripts/activate
```

Repeat the `.venv/Scripts/activate` command to re-activate the virtual environment, whenever you open a new terminal to build the documentation.

- Open a Command Prompt. (Run as Administrator)
- Enter the `blender-manual` folder which was just added by `git clone`:

  ```
  cd C:\blender-manual
  ```

- Inside that folder is a file called `requirements.txt` which contains a list of all the dependencies we need. Install all the dependencies using Python's `pip` command:

  ```
  pip install -r requirements.txt
  ```

- If all goes well, you should see the following message when it is finished:

  ```
  Successfully installed Jinja2 MarkupSafe Pygments Sphinx
  docutils sphinx-rtd-theme Cleaning up...
  ```

During the setup, some warnings may be shown, but do not worry about them. However, if any errors occur, they may cause some problems.

Note

Every now and then you may want to make sure your dependencies are up to date using:

```
pip install -r requirements.txt --upgrade
```

Building the Manual

Converting the RST-files into pretty HTML pages is simple.

Open a terminal or Command Prompt in the `~/blender-manual` directory and simply run:

```
make
```

Tip

On Microsoft Windows you can simply open the `make.bat` file to easily run the command without having to open the Command Prompt and typing commands.

This is the command you should use when building the docs, however, other commands are available by typing `make help`. This command will convert the RST-files into HTML pages and automatically open your default web browser to view the result. The command will continue to run and watch for changes made to the RST-files and refresh the HTML pages as necessary.

Note

The converted pages can also be viewed manually by browsing the build directory: `~/blender-manual/build/html`. For example to open the home page, open `build/html/index.html` to read the manual.

The building process may take several minutes the first time (or after any major changes), but for subsequent changes it should only take a few seconds.

Editing the Manual

You can modify the manual by editing local text files. These files are kept in sync with those online via a repository, based on this the server will update the online manual.

The manual is written in the reStructuredText (RST) markup language and can be edited using a plain text editor. For a local preview, you convert (build) the manual source files from RST into HTML web pages.

Update

Firstly, make sure that your local copy of the manual is up to date with the online repository using:

```
make update
```

Writing

You can now edit the documentation files, which are the `.rst` files inside the `manual` folder using a text editor of your choice.

Be sure to check the Writing Style Guide for conventions and Markup Style Guide to learn how to write in the reStructuredText markup language.

Happy writing!

Bigger Changes

If you are going to add or overhaul a section, be sure to check carefully that it does not already exist. In some places, the docs are so disorganized that sections may be duplicated or in a strange location. In the case that you find a duplicate or out of place section, create an issue explaining the issue, and optionally include a revision (actual changes).

Before you make any edits that are not simple and plainly justified (for example, moving folders around), you should verify with a manual maintainer that your contribution is along the community's vision for the manual. This ensures the best use of your time and good will as it is otherwise possible that, for some reason, your changes will conflict and be rejected or need time-consuming review. For example, another person may be already working on the section you wish to change, the section may be scheduled for deletion or to be updated according to a planned change to Blender.

Communicating early and frequently is the key to have a productive environment, to not waste people's effort and to attain a better Blender manual as a result.

Getting Help/Answers

If you are in doubt about functionality that you wish to document, you should pose your questions to the Blender developers responsible for that area or ask at the unofficial user support channel in the `#docs` or `#blender-coders` channel in Chat.

Blender itself has a system of module owners with developer and artist members who are responsible for the design and maintenance of the assigned Blender areas. See the module pages for more information.

Preview

To view your changes, build the manual as instructed. Keep in mind that you can also build only the chapter you just edited to view it quickly. Open the generated `.html` files inside the `build/html` folder using your web browser, or refresh the page if you have it open already.

Upload

When you are happy with your changes, you can upload them, so that they will be in the online manual. At first, this is done by submitting patches so that someone can review your changes and give you feedback. After, you can commit your changes directly. This process is described in detail in the next section.

Pull Requests

This page describes the tools used for code contribution and review.

Reviews are a key measure to assure changes are of a good quality. They help preventing bugs, design consistencies, or potential maintenance problems. And having your work reviewed also generally keeps you on your toes.

Note

Writers who have been given commit access can commit to the main repository without needing to fork the repository.

See Commit Guidelines if this applies to you.

One Time Setup

This assumes you have the Blender manual repository already checked out on your computer, following the install instructions.

Fork

1. Go to Blender repository and click the Fork button.
2. Confirm the fork with the default settings.
3. Now you will have to add your personal fork as a remote in your local git repository. Click *SSH* to see the correct URL, and then add it like this:

   ```
   git remote add me git@projects.blender.org:
   <USERNAME>/blender-manual.git
   ```

Note

In order to push to the fork repository, you need an SSH key. If you don't already have the file `~/.ssh/id_rsa.pub`, there's a simple command to generate such keys which works on Linux, macOS, and in Git Bash on Windows:

```
ssh-keygen
```

This command will generate a private key id_rsa and a public key id_rsa.pub in `~/.ssh`. The private key must never be shown or sent to anyone else to avoid compromising your account, but the public key is safe to share.

The contents of `~/.ssh/id_rsa.pub` can be copied and pasted into the account settings on projects.blender.org, after clicking "Add Key". Any name for the SSH key is ok.

Workflow

The workflow for working with pull requests can be found in the Blender Developer's Documentation.

Note, some text in the above guideline is focused on the main Blender repository, however, the workflow is the same for any Blender project.

Guidelines for Reviewers

- The pull request text should be usable as the git commit message (see the guidelines for details).
- Be explicit when some changes are to be addressed before committing, without the need for a review iteration.
- If the pull request is not approved the author is expected to make another iteration.

- If the change needs agreement on the design task first, put the pull request on hold by adding a `WIP:` prefix in the title, indicating the author considers the pull request not ready to be merged. No review is expected unless the author specifically asks for it.
- Writers are expected to reply to pull requests in 3 working days.
- Add relevant modules/projects to the labels.
- Encourage new writes to do review, it's a good way to learn and important to grow the project.

Tips

- To get the patch file, add .patch to the end of the URL of the pull request. Example:

  ```
  https://projects.blender.org/blender/blender-manual/pulls/104892.patch
  ```

- Checkout a pull request into a detached head (not leaving behind a branch). Example:

  ```
  git fetch -q origin +refs/pull/104892/head: ; git checkout -qf FETCH_HEAD
  ```

Todo List

This page provides a list of changes that need to be made to the manual. This is a great place for new contributors to start but also check the documentation workboard.

Writing Style Guide

Primary Goals

The main goals for this manual are as follows:

User Focused

The manual is written for people educated in computer graphics, who understand the basics of 3D and/or know other 3D software. While some areas of computer graphics are highly technical, this manual shall be kept understandable by non-technical users.

Complete

The manual provides detailed functional description of all features, tools and options in Blender. While there is a canonical source of truth for each of Blender's key areas, this does not mean we have to document every small detail. The manual should provide information on what a feature is, how to use it, and its purpose. More background information should be provided when necessary to give deeper understanding of a 3D pipeline.

Concise

Computer graphics is an incredibly interesting field, there are many rules, exceptions to the rules, and interesting details. Expanding into details can add unnecessary content, so keep the text concise and relevant to the topic at hand.

Maintainable

Keep in mind that Blender has frequent releases, so try to write content that will not have to be redone the moment some small change is made. This also helps a small documentation community to maintain the manual.

Content Guidelines

In order to maintain a consistent writing style within the manual, please keep this page in mind and only deviate from it when you have a good reason to do so.

Rules of thumb:

- Spell checking is *strongly* recommended.
- Use American English (e.g: modeling and not modelling, color and not colour) also for formatting numbers (e.g: 2,718.28 and not 2 718,28).
- Take care about grammar, appropriate wording and use simple English.
- Keep sentences short and clear, resulting in text that is easy to read, objective and to the point.
- Including why or how an option might be useful is a good idea.
- If you are unsure about how a feature works, ask someone else or find out who developed it and ask them.

To be avoided:

- Avoid writing in first person perspective, about yourself or your own opinions.

- Avoid weasel words and being unnecessarily vague, e.g:

  "Reloading the file will probably fix the problem"
  "Most people do not use this option because …"

- Avoid including specific details such as:

  "Blender has 23 different kinds of modifiers."
  "Enabling previews adds 65536 bytes to the size of each blend-file (unless it is compressed)."

  These details are not useful for users to memorize and they become quickly outdated.

- Avoid documenting bugs.

  Blender often has 100's of bugs fixed between releases, so it is not realistic to reference even a fraction of them from the manual, while keeping this list up to date.

Issues that are known to the developers and are not going to be resolved before the next release can be documented as *Known Limitations*. In some cases, it may be best to include them in the troubleshooting section.

- Avoid product placements, i.e. unnecessarily promoting software or hardware brands. Keep content vendor-neutral where possible.

- Avoid technical explanations about the mathematical/algorithmic implementation of a feature if there is a simpler way to explain it.

  (E.g. explaining how mesh smoothing algorithms work is unnecessary, but the blending types of a Mix node do need a mathematical explanation.)

- Avoid repetition of large portions of text. Simply explain it once, and from then on refer to that explanation.

  For general terminology, consider defining a `:term:` in the glossary.

- Avoid enumerations of similar options, such as listing every preset or every frame rate in a select menu.

  Their contents may be summarized or simply omitted. – Such lists are only showing what is already *obvious* in the interface and end up being a lot of text to read and maintain.

- Avoid documenting changes in Blender between releases, that is what the release notes are for. We only need to document the current state of Blender.

- Unless the unit a value is measured in is obscure and unpredictable, there is no need to mention it.

- Do not simply copy the tooltips from Blender. – People will come to the manual to learn *more* than is provided by the UI.

As a last resort you can add comment (which is not shown in the HTML page, but useful for other editors):

```
.. TODO, how does this tool work? ask Joe Blogg's.
```

Glossary

This section is specifically about the Glossary section, where we define common terms in Blender and computer graphics.

Rules of thumb:

- Define the term before providing any further information.
- Avoid using constructs such as “it is” or “xyz is” before the definition.
- Avoid repeating the term immediately or using it in the definition.
- Avoid adding terms not found in Blender’s interface or the manual.
- Avoid overly long entries. If an explanation of a complex term is needed, supplement with external links.
- Avoid duplicating documentation; if explaining the term is the primary focus of another section of the manual (e.g. if the term is the name of a tool), either just link to that section, or avoid creating a glossary entry entirely.
- URL references are to be added at the end, formatted as follows, e.g:

  ```
  See also `OpenGL <https://en.wikipedia.org/wiki/OpenGL>`__
  on Wikipedia.
  ```

Examples

This entry:

```
Displacement Mapping
   Uses a grayscale heightmap, like Bump Mapping,
```

```
   but the image is used to physically move the vertices of the
mesh at render time.
   This is of course only useful if the mesh has large amounts
of vertices.
```

Would be written like this instead, putting a definition first:

```
Displacement Mapping
   A method for distorting vertices based on an image.
   Similar to Bump Mapping, but instead operates on the mesh's
actual geometry.
   This relies on the mesh having enough geometry.
```

This entry:

```
Doppler Effect
   The Doppler effect is the change in pitch that occurs
   when a sound has a velocity relative to the listener.
```

Would be written more like this, avoiding the immediate repetition of the term:

```
Doppler Effect
   Perceived change in pitch that occurs
   when the source of a sound is moving relative to the
listener.
```

This entry:

```
Curve
   It is a class of objects.
   In Blender there are Bézier curves and NURBS curves.
```

Would be written more like this, avoiding the “it is”:

```
Curve
   A type of object defined in terms of a line interpolated
between Control Vertices.
   Available types of curves include Bézier and NURBS.
```

Markup Style Guide

This page covers the conventions for writing and use of the reStructuredText (RST) markup syntax.

Conventions

- Three space indentation.
- Lines should be less than 120 characters long.
- Use italics for button/menu names.

Other loose conventions:

- Avoid Unicode characters.
- Avoid heavily wrapped text (i.e. sentences can have their own lines).

Headings

```
#################
  Document Part
#################

****************
Document Chapter
****************

Document Section
================

Document Subsection
-------------------

Document Subsubsection
^^^^^^^^^^^^^^^^^^^^^^
```

```
Document Paragraph
""""""""""""""""""
```

Note

Parts should only be used for contents or index pages.

Note

Each `.rst` file should only have one chapter heading (*) per file.

Text Styling

See the [overview on ReStructuredText]() for more information on how to style the various elements of the documentation and on how to add lists, tables, pictures and code blocks. The [Sphinx reference]() provides more insight additional constructs.

The following are useful markups for text styling:

```
*italic*
**bold**
``literal``
```

Interface Elements

- `:kbd:`LMB`` – keyboard and mouse shortcuts.
- `*Mirror*` – interface labels.
- `:menuselection:`3D Viewport --> Add --> Mesh --> Monkey`` – menus.

Code Samples

There is support for syntax highlighting if the programming language is provided, and line numbers can be optionally shown with the `:linenos:`

option:

```
.. code-block:: python
   :linenos:

   import bpy
   def some_function():
       ...
```

Images

Figures should be used to place images:

```
.. figure:: /images/interface_window-system_splash_current.png

   Image caption.
```

For consistency, and since it would be good to ensure screenshots are all a similar size when floated next to text, writers should take screenshots in the following manner:

1. Prepare the area you would like to capture making sure to use the default theme and setting. (In some cases you may not want to use the default settings e.g. if some options are hidden behind a checkbox.)
2. Zoom to the maximum zoom level (hold `NumpadPlus` or `Ctrl-MMB` or similar).
3. Zoom out eight zoom levels (`NumpadMinus` – eight times).
4. In some cases you will want to leave a small margin around the thing you are trying to capture. This should be around 30px but does not have to be exact.

This can be applied to several parts of the interface but might not work for all cases.

Files

No Caps, No Gaps
    Lower case filenames underscore between words.

Sort Usefully

Order naming with specific identifiers at the end.

Format

Use `.png` for images that have solid colors such as screenshots of the Blender interface, and `.jpg` for images with a high amount of color variance, such as sample renders and photographs.

Do not use animated `.gif` files, these are hard to maintain, can be distracting and are usually large in file size. Instead use a video if needed (see Videos below).

Location

Place the image in the `manual/images` folder. Use no other subfolders.

Naming

For naming files use underscores to separate chapters and sections, and use dashes to separate sections that are two or more words. So for image files should look like: `chapter_subsection_sub-subsection_id.png`, e.g:

- `interface_splash_current.png`
- `interface_undo-redo_last.png`
- `interface_undo-redo_repeat-history-menu.png`

Do not use special characters or spaces!

Usage Guides

- Avoid specifying the resolution of the image, so that the theme can handle the images consistently and provide the best layout across different screen sizes.

- When documenting a panel or section of the UI, it is better to use a single image that shows all of the relevant areas (rather than multiple images for each icon or button) placed at the top of the section you are writing, and then explain the features in the order that they appear in the image.

Note

It is important that the manual can be maintained long term, UI and tool options change so try to avoid having a lot of images (when they are not especially necessary). Otherwise, this becomes too much of a maintenance burden.

Videos

Videos can be embedded from Blender's self-hosted PeerTube instance which can be found at video.blender.org. To embed a video using the following directive:

```
.. peertube:: ID
```

The `ID` is found in the video's URL, e.g:

The ID for `https://video.blender.org/videos/watch/47448bc1-0cc0-4bd1-b6c8-9115d8f7e08c` is `47448bc1-0cc0-4bd1-b6c8-9115d8f7e08c`.

To get a new video uploaded, contact a Documentation Project Administrator or include the uploaded video in your Pull Request description.

Usage Guides

- Avoid adding videos that rely on voice or words, as this is difficult to translate.
- Do not embed video tutorials as a means of explaining a feature, the writing itself should explain it adequately. (Though you may include a link to the video at the bottom of the page under the heading `Tutorials`).

Useful Constructs

- `|BLENDER_VERSION|` – Resolves to the current Blender version.
- ``:abbr:`SSAO (Screen Space Ambient Occlusion)` `` – Abbreviations display the full text as a tooltip for the reader.
- ``:term:`Manifold` `` – Links to an entry in the Glossary.

Cross References and Linkage

You can link to another document in the manual with:

```
:doc:`The Title </section/path/to/file>`
```

To link to a specific section in another document (or the same one), explicit labels are available:

```
.. _sample-label:

[section or image to reference]

Some text :ref:`Optional Title <sample-label>`
```

Linking to a title in the same file:

```
Titles are Targets
==================

Body text.

Implicit references, like `Titles are Targets`_
```

Linking to the outside world:

```
`Blender Website <https://www.blender.org>`__
```

Context Sensitive Manual Access

It is possible to link to a specific part of the manual from in Blender by opening the context menu (right click) of a property or operator and selecting *Online Manual*. In order for this to work, this needs to be accounted for in the documentation. To link a property or operator to a specific part of the manual you need to add an external reference link tag whose ID matches Blender's RNA tag. The easiest way to find out what the tag for a property is to open the context menu of the property/operator and select *Online Python Reference* to extract the tag from the URL. Some examples of how this looks in the RST document are given below:

```
.. _bpy.types.FluidDomainSettings.use_fractions:

Fractional Obstacles
   Enables finer resolution in fluid / obstacle regions (second
order obstacles)...

   .. _bpy.types.FluidDomainSettings.fractions_distance:

   Obstacle Distance
      Determines how far apart fluid and obstacles are...
```

For an operator:

```
.. _bpy.ops.curve.subdivide:

Subdivide
=========
```

Icons

Blender's icons can be included as inline text using:

```
`:bl-icon:`<icon_name>`
```

Further Reading

To learn more about reStructuredText, see:

[Sphinx RST Primer]
    Good basic introduction.

[Docutils reStructuredText Reference]
    Links to reference and user documentation.

Commit Guidelines

Access to directly submit changes is limited to people with commit access to the repository. Once you are provided with commit access you can start committing directly instead of creating a patch file.

You can make commits from your Git client or using the Git command line tool. The following command will create a commit and send it to the central repository:

```
git commit -m "This is what I did"
git push
```

If you leave out `-m "message"`, you will be prompted to type the message in a text editor.

Tip

You should make sure you are always on the latest revision before committing. You may not be able to commit directly if there are conflicting changes in the latest revision.

To avoid this update your local repository before committing (run `make update`).

See also

Blender's Git usage guide

See also

See release branch for documentation on how to make commits to a specific release branch and how to create merge commits.

Writing a Good Commit Message

When making changes to the manual that directly relate to a specific commit (change) in Blender, it is helpful to make the title of the commit the same as the commit made to Blender. It is requested that you include the commit hash of the commit made to the Blender source code.

For example, the commit rBM8473 includes a descriptive indicative of the changes made along with the hash `rBa71d2b260170`. The hash can be extracted from the URL provided in the Documentation task for a specific upcoming release.

Other more general changes do not have to follow the above policy however, it is still important to make the description clear about what changes you made and why. It can be helpful to prefix the commit title with a prefix word such as `Cleanup:` or `Fix:` when you are making general cleanups or fixes respectively.

Writing good commit messages helps administrators keep track of changes made and ensures all new features are properly documented.

Templates

The following guide provides patterns for interface elements and directories.

Operator Menus

Each operator should receive its own heading or page based on the length of the content. At the start should be a reference admonition documenting the context of the operator:

```
.. admonition:: Reference
   :class: refbox

   :Mode:      Edit Mode
   :Menu:      :menuselection:`Curve --> Snap`
   :Shortcut:  :kbd:`Shift-S`
```

Panels

Panels should be documented by their own heading, nested panels should use decreasing heading levels. Each panel could have its own page based on the length of documentation and/or the amount of panels. Expanded menus that toggle what properties are presented to the user should be treated like subpanels:

```
Panel Title
===========

Nested Panel Title
------------------
```

Properties

Properties should be documented using definition lists. Properties that are hidden based on other properties should used nested definitions:

```
Property
   Property description.

   Hidden Property
      Hidden property description.
```

Select menus should be documented using the following syntax:

```
Menu Label
   General description of the menu.

   :Menu Item: Menu Item Definition.
   :Menu Item: Menu Item Definition.
   :Menu Item: Menu Item Definition.
```

Nodes

Nodes should always have three headings inputs, properties and outputs with a note of absence if the node has none. At the end of the page can be an optional example(s) section:

```
**********
World Node
**********

.. figure:: /images/render_shader-nodes_output_world_node.png
   :align: right

   The World node.

Introduction and general use case(s).


Inputs
======

This node has no inputs.
```

```
Properties
==========

This node has no properties.


Outputs
=======

This node has no outputs.


Example
=======
```

Directory Layout

Sections should be generally structured as follows:

- `directory_name/`
  - `index.rst` (contains links to internal files)
  - `introduction.rst`
  - `section_1.rst`
  - `section_2.rst`

For example:

- `rendering/`
  - `index.rst`
  - `cycles/`
    - `index.rst`
    - `introduction.rst`
    - `materials/`
      - `index.rst`
      - `introduction.rst`
      - `volumes.rst`

The idea is to enclose all the content of a section inside of a folder. Ideally every section should have an `index.rst` (containing the TOC for that section) and an `introduction.rst` (introducing) to the contents of the section.

Table of Contents

By default, a table of contents should show two levels of depth:

```
.. toctree::
   :maxdepth: 2

   introduction.rst
   perspective.rst
   depth_of_field.rst
```

Maintenance

Adding/Removing/Moving Files

When RST-files are added or removed the corresponding locale files are added or removed automatically by the update script. However, if files need to be moved please use this Python script:

```
python tools/utils_maintenance/rst_remap.py start
```

RST-files can then be freely moved and the remap script will move the locale file after:

```
python tools/utils_maintenance/rst_remap.py finish
```

It is best to avoid moving/renaming files as this breaks URLs and without this script translators will lose all their work in these files. Please ask an administrator if you think something should be renamed/moved.

Note

This script also works for image file names.

Release Checklist

- Create a release branch (`blender-3.2-release/`)
- Update the splash image: `interface_splash_current.png` in the release branch.
- Increase the `conf.py: blender_version` variable in the trunk version.

Contribute

On this page French (`fr`) is used for examples. However, it can be replaced with other languages codes. So, be sure to change the `/fr` suffixes in this guide to the language you are translating!

To see which languages are currently available, you can check the online interface, or browse the underlying git repository.

Simple Contribution

The preferred way to contribute to the translation effor is the use the web-based interface, currently a Weblate instance.

Simple enhancement suggestions can be contributed by any user, even without loging in. Suggestions will be reviewed by the translating team before they get published.

Weblate also comes with new helping tools to improve coherance of translations, like the glossary.

Advanced Operations

If for some reasons the web-based translation interface does not work well for you, you can still download PO file from it, and upload back it later.

Warning

You will have to deal with potential conflicts yourself if some update happened in the mean time. Direct commit to the git repository for translations is not possible anymore.

Note

There is a known issue with the current tool behind the web interface, which will make heavy processing like upload and integration of a PO file take several minutes, with the web page staying in refresh mode for the whole time. If it takes more than ten minutes, it will even apparently fail with a server timeout error message. There is usually no actual problem though, so no need to re-try uploading the PO file then, refreshing the page after a few minutes should be enough to see the contribution in the web interface.

Note

First of all, it is assumed that you have the manual already building. If you have not done this already go back too the Getting Started section.

Installing

Language Files

From the directory containing your checkout of the manual run:

```
make checkout_locale
```

You will be prompted to type in the language folder you want to download. In the case of this example we will use `fr`. Pressing `Return` will confirm this selection.

It will take a few minutes to download but once complete it will create a `locale/fr` subdirectory.

You should have a directory layout like this:

```
blender-manual
   |- locale/
   |  |- fr/
   |  |  |- LC_MESSAGES/
   |- manual/
```

Note

When running Git from the command line (such as updating), you will need to change directory to `locale` first rather than the `blender-manual` directory.

The PO language files themselves can also be downloaded from the web interface, `Files` menu, on each dedicated language page of the `Manual` comnponent.

A PO Editor

To make edit the PO files you will need to install a PO editor. We recommend that you use Poedit, however any PO editor will do.

Note

For Linux users, you will have to check with your distribution's software center for a version of Poedit. This editor is only a recommendation. There are others, such as Kate and Kwrite, that could offer syntax highlighting and basic tools for text editing, e.g. letter case transposes. Other platforms can use some text editors supporting the syntax highlighting for PO files, or allowing you to create a custom one (such as Notepad++ on Windows).

Building with Translations

Building

Now you can build the manual with the translation applied:

On Linux and macOS run:

```
make -e BF_LANG=fr
```

On Windows run:

```
set BF_LANG=fr
make html
```

Now you will have a build of the manual with translations applied.

Editing Translation Files

Now you can edit the PO translation files, in the `LC_MESSAGES` folder you have two files:

- `blender_manual.po` – This is the main translation file that you will be editing.
- `sphinx.po` – This translation file is much smaller and contains translations for the website theme.

To edit these files open them up in your translation editor, i.e. Poedit. Once in your editor you will see a list of texts, each of these items represent some part of the user manual. You may need to adjust your editor to sort the list in a way that makes sense for example "by source".

You can now select an untranslated string and your editor will have an input box to add the translation. The modified `.po` files can now be submitted back to the web-based interface.

Tip

Make sure that you Building with Translations to catch any syntax errors you may make while translating. These errors will be displayed as warnings while building the manual.

Maintenance

Keeping Track of Fuzzy Strings

When the manual is updated, those translations which are outdated will be marked as fuzzy. To keep track with that, you can use a tool we created for that task.

You can do this by running:

```
make report_po_progress
```

This will only give a quick summary however, you can get more information by running:

```
python tools/translations/report_translation_progress.py
locale/fr/
```

You should get a list of all the files with information about the number of empty and fuzzy strings. For more options see:

```
python tools/translations/report_translation_progress.py --help
```

Updating PO Files

As the original manual changes, the templates will need updating. Note, doing this is not required, as administrator usually update the files for all languages at once. This allows all languages to be on the same version of the manual. However, if you need to update the files yourself, it can be done as follows:

```
make update_po
```

The updated templates can then be committed to the repository.

See also

A guide how to add a new language can be found in the Adding a Language.

Style Guide

This page covers conventions concerning the translations.

Note

- We expect our readers to use the English version of Blender, not a translated one.
- The translations are licensed under the same License as the original.

Should I Translate... ?

Maybe

Hyperlinks
Can be translated, but only as an addition, not as a replacement. See also Adding Text.

Technical Terms
Only translate these, when the localized expression is common! See also Technical Terms.

Text you are not sure you understood
Simply mark the text as fuzzy and/or add a comment. The next translator might understand it.

Never

Images
You probably will not find the original scene if it is a screenshot of a file and it is too much load on the server (and too much work for you).

Menu and button names

We expect our readers to use the English UI.

Text you do not understand

Do not translate it! It will do more harm than good!

Technical Terms

In general, the technical terms used in computer graphics are quite new or even downright neologisms invented for the needs, so they do not always have a translation in your language. Moreover, a large part of Blender users use its English interface.

As a result, unless a term has an evident translation, you should preferably use the English one, *putting it in italic*. You can then find a translation for it, which you will use from times to times (e.g. to avoid repetitions…). This is also valid in the other way: even when a term has a straightforward translation, do not hesitate to use its English version from times to times, to get the reader used with it…

If a term is definitively not translatable, simply use the English one, but make sure its glossary entry is translated.

In the glossary, the English term is written first (to maintain alphabetic order) with the translated entry following in parenthesis, when appropriate.

Adding Text

Generally, **you should always translate exactly what is in the text**, and avoid providing updates or extra information.

But sometimes that is necessary, for example when talking about the manual itself: To a foreign reader it is not clear, that they can contribute English text only, whereas this is obvious to an English reader.

In these (rare) cases you can and should provide extra information.

Keeping Pages Up To Date

When the manual is updated, those translations which are outdated will be marked as fuzzy. To keep track with that, you can use a tool we created for that task, see How to install it.

Not Found (404)

It seems the page you are trying to find has been either moved or deleted.
You can try one of the following things:

Search

License

Blender itself is released under the GNU General Public License. More info blender.org/about/license.

Except where otherwise noted, the content of the Blender Manual is available under a Creative Commons Attribution-ShareAlike 4.0 International License or any later version. Excluded from the CC-BY-SA are also the used logos, trademarks, icons, source code and Python scripts.

Please attribute the “Blender Documentation Team” and include a hyperlink (online) or URL (in print) to manual. For example:

```
The Blender 4.3 Manual
by the Blender Documentation Team
is licensed under a CC-BY-SA v4.0.
```

See Best practices for attribution for further explanation.

It means, that when contributing to the manual you do not hold exclusive copyright to your text. You are, of course, acknowledged and appreciated for your contribution. However, others can change and improve your text in order to keep the manual consistent and up to date.

If you have questions about the license, feel free to contact the Blender Foundation: foundation (at) blender (dot) org

Previous versions of the Blender Manual are made available under an Open Content License v2.26 – v2.77.

Adding a Language

Preparations

If the language you want to translate has not been started by someone else already and you wish to create a set of new files for the desired language, say ‘fr’ (French), then you must first use the environment you have created, as guided in Getting Started, in particular Installing Dependencies and Building the Manual sections.

This will give you a foundation environment for:

- Creating a new set of translation language from English source.
- Perform `make` command to turn translated texts in po files into html files for testing locally.
- Update changes in English texts which have been added by other contributors.

Below examples show the process to create a new set of files for French, language code `fr`, on Linux platform. Other platforms might vary slightly but should be mainly the same.

1. Create a Blender ID if you have not done so already.
2. Log into projects.blender.org and Create an Issue requesting for commit access in order to transfer changes to the central repository of the translation team.
3. Open an instance of a console application.
4. Change the current working directory to the directory of `blender-manual`, where the instance of `Makefile` resides.

Trying the Make Process to Create HTML Files in English

1. Ensure the previous instance of `build` directory is removed, if any exists:

   ```
   make clean
   ```

2. Convert all the `rst` files into `pot` translation files:

   ```
   make gettext
   ```

3. Create `html` files:

   ```
   make html
   ```

4. After this, you can actually view the created html files locally by opening the `blender-manual/build/html/index.html` file.

Creating the Language Entry in the HTML Menu

1. Create an entry for the language in the html menu by opening file `./build_files/theme/js/version_switch.js` (assuming you are at the `blender-manual` subdirectory).

2. Find the table for the languages in `var all_langs = {..};`.

3. Enter the entry: `"fr": "Fran&ccedil;ais",`, (`"fr": "Français"`). (Notice the Unicode characters.)

4. Commit the updated file:

   ```
   git add ./build_files/theme/js/version_switch.js
   git commit -m "HTML: Add French to language menu"
   ```

5. Push your changes to the upstream repository:

   ```
   git push
   ```

Generating the Set of Files for the Target Language

1. Check out the current translation repository using the command:

   ```
   git clone https://projects.blender.org/blender/blender-
   manual-translations.git locale
   ```

   This will download all language sets available in the repository into the `locale` directory of your drive. You can go to the `locale` directory to see the hidden subdirectory `.git` within it, together with directories of languages. You'll need to add your own set of files for the language you are trying to translating to.

2. From the `blender-manual` directory to generate a set of files for `fr` language:

   ```
   make update_po
   ```

   These files are still in English only, with all `msgstr` entries blank.

3. Submit new set of files to the central repository:

   ```
   cd locale
   git add fr
   git commit -m "Initial commit language set of files for
   French"
   ```

Tip

- It is recommended you make two environment variables for these directories, in the `.bashrc` to make it more convenient for changing or scripting batch/shell commands for the process of translation and reviewing results:

  ```
  export BLENDER_MAN_EN=$HOME/<directory to make file
  directory above>/blender-manual
  export BLENDER_MAN_FR=$BLENDER_MAN_EN/locale
  ```

- Newly generated files will contain some placeholders for authors and revision dates etc. If you find the job of replacing them repetitive, make use of the script `change_placeholders.sh` in the subdirectory `~/blender-manual/tools/util_maintenance`, make a copy of that to your local `bin` directory and replace all values that were mentioned in the file with your specific details, then after each change to a file, you would do following commands to update the file with your personal details, revision date and time, plus generating the html files for your language, which you can view using your Internet browser:

```
$HOME/bin/change_placeholders.sh $BLENDER_MAN_FR
make -d --trace -w -B -e SPHINXOPTS="-D language='fr'"
2>&1
```

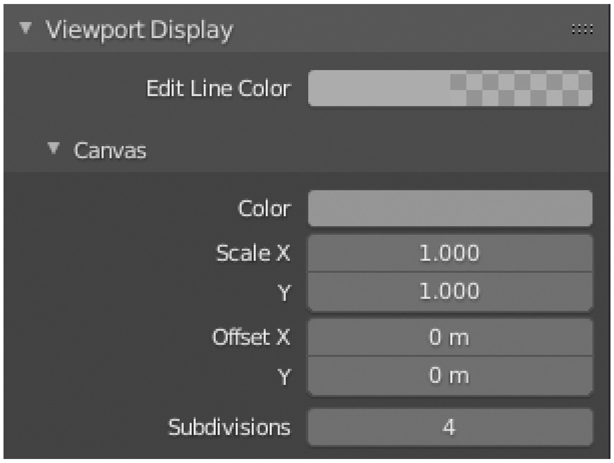
Getting Started

- About Blender
    - Who uses Blender?
    - Key Features
    - Further Reading
- Installing Blender
    - System Requirements
    - Download Blender
    - Installation Guides
- Configuring Blender
    - Introduction
    - Configuring Peripherals
    - Defaults
- Help System
    - Tooltips
    - Context-Sensitive Manual Access
    - Help Menu

Viewport Display

Display settings for Edit Lines in *Edit Mode* and *Sculpt Mode*.

Edit Line Color
: Sets the color of the Edit Lines.



Viewport Display

Edit Line Color

Canvas

Color

Scale X 1.000

Y 1.000

Offset X 0 m

Y 0 m

Subdivisions 4

Viewport Display panel.

Canvas

In 3D space sometimes is difficult to assess on which plane are you drawing. The Canvas is a display overlay helper that shows a grid at the current *Drawing Plane*. You can enable the Canvas visualization in the Viewport Overlays.

See Drawing Plane for more information.

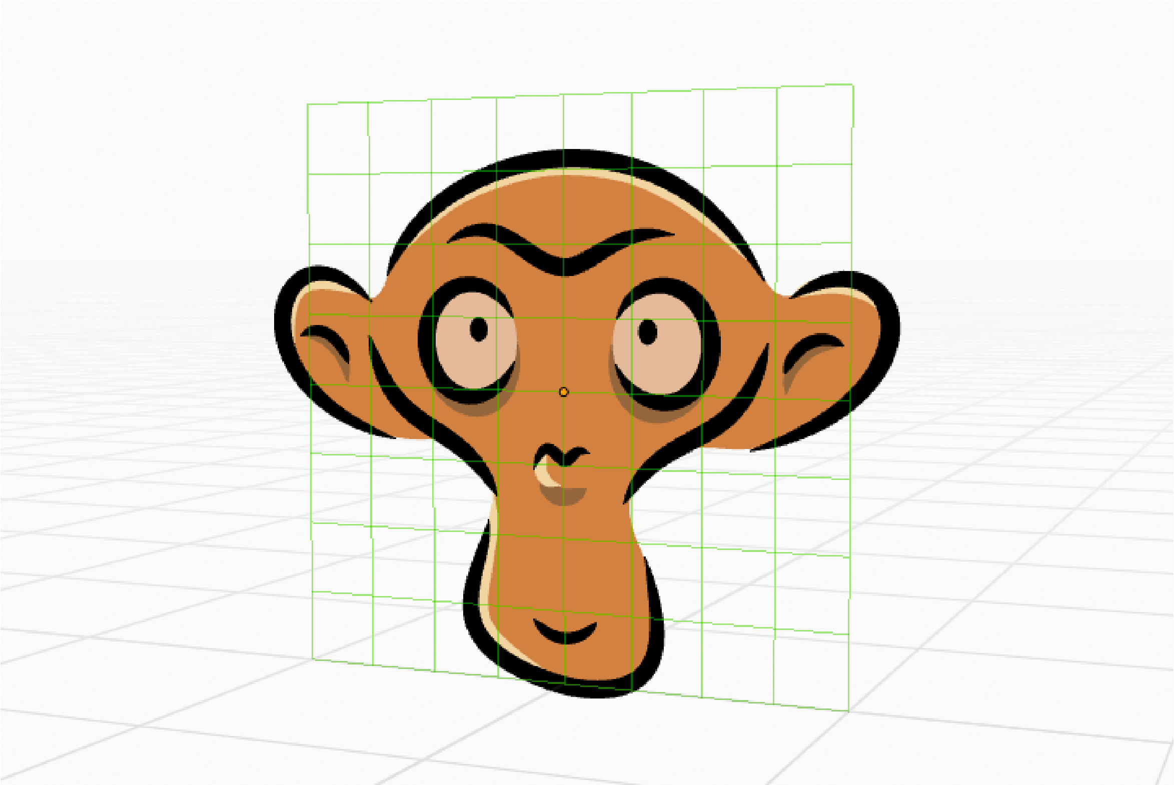
Color
: Color of the Canvas grid lines.

Scale X/Y
: Defines the X and Y scale of the Canvas.

Offset X/Y
: Sets the Canvas position offset from the object’s origin.

Subdivisions

Specifies the number of subdivisions to use for the grid.



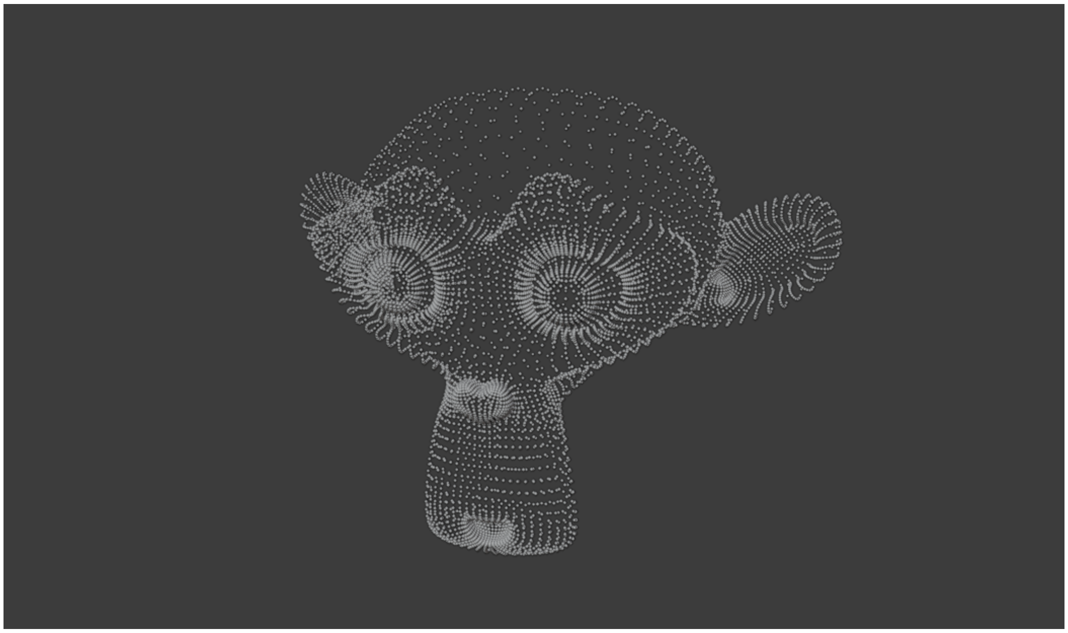
Canvas example on the XZ drawing plane using a green color grid.

Important

This feature is currently experimental and not available in current releases.

Point Cloud

Point clouds can be used to represent 3D scans and in the future can represent particles. Each point can store data in a set of Attributes.



Example of a monkey object represented as a point cloud.

Properties

Attributes

The *Attributes* panel contains different point cloud characteristics such as the position and size of points. Use the List View to manage attributes.

Attribute Types

See also

See Built-In Attributes for information about common attributes.

Name	Type	Domain	Notes
`position`	*Vector*	*Point*	Built-in attribute describing vertex or point locations, in the modifier object's transform space.
`radius`	*Float*	*Point*	The radius of each point.
`color`	*Color*	*Point*	The color of each point.
`id`	*Integer*	*Point*	A unique identifier given to each particle.
`velocity`	*Vector*	*Point*	The speed and direction that the particle is traveling.

Custom Attributes

Custom attribute can be given to particles to hold a custom characteristic.

Name

The name of the attribute.

Data Type

The type of data to store in the attribute.

Float::	Floating-point value
Integer::	32-bit integer
Vector::	3D vector with floating-point values
Color::	RGBA color with floating-point precision
Byte Color::	RGBA color with 8-bit precision
String::	Text string

Domain

The type of element the attribute is stored in. Currently, attributes can only be stored per *Point*.

Custom Properties

See the Custom Properties page for more information.

Editing

Currently, not much can be done with point clouds; however, they can be converted to/from meshes.

Line Style Modifiers

Color

- Along Stroke
- Crease Angle
- Curvature 3D
- Distance from Camera
- Distance from Object
- Color Modifiers
    - Along Stroke
    - Crease Angle
    - Curvature 3D
    - Distance from Camera
    - Distance from Object
    - Material
    - Noise
    - Tangent
- Material
- Noise
- Tangent

Alpha

- Along Stroke
- Crease Angle
- Curvature 3D
- Distance from Camera
- Distance from Object
- Alpha Modifiers
    - Along Stroke
    - Crease Angle
    - Curvature 3D
    - Distance from Camera

  - Distance from Object
  - Material
  - Noise
  - Tangent
- Material
- Noise
- Tangent

Thickness

- Along Stroke
- Calligraphy
- Crease Angle
- Curvature 3D
- Distance from Camera
- Distance from Object
- Thickness Modifiers
  - Along Stroke
  - Calligraphy
  - Crease Angle
  - Curvature 3D
  - Distance from Camera
  - Distance from Object
  - Material
  - Noise
  - Tangent
- Material
- Noise
- Tangent

Geometry

- 2D Offset
- 2D Transform
- Backbone Stretcher
- Bézier Curve

- Blueprint
- Guiding Lines
- Geometry Modifiers
    - 2D Offset
    - 2D Transform
    - Backbone Stretcher
    - Bézier Curve
    - Blueprint
    - Guiding Lines
    - Perlin Noise 1D
    - Perlin Noise 2D
    - Polygonization
    - Sampling
    - Simplification
    - Sinus Displacement
    - Spatial Noise
    - Tip Remover
- Perlin Noise 1D
- Perlin Noise 2D
- Polygonization
- Sampling
- Simplification
- Sinus Displacement
- Spatial Noise
- Tip Remover

On macOS graphics drivers are built into the operating system and the only way to get newer drivers is to upgrade macOS as a whole to the latest version.

Blender uses of OpenGL for the 3D Viewport and user interface. The graphics card (GPU) and driver have a big impact on Blender’s behavior and performance.

This section lists possible solutions for graphics glitches, problems with EEVEE and Cycles, and crashes related to your GPU.

Drivers

Upgrading to the latest graphics drivers often solves problems. Newer drivers have bug fixes that help Blender function correctly.

Laptops

Laptops often have two GPUs for power saving purposes. One slower onboard GPU (typically Intel) and one faster dedicated GPU for a better performance (AMD or Nvidia).

For the best performance the dedicated GPU should be used for Blender. Which GPU to use for which application can be configured in your graphics driver settings.

If there is a graphics glitch or crash specific to the onboard GPU, then using the dedicated GPU can help avoid that. Or vice versa, if the dedicated GPU causes issues, then using the onboard graphics can help.

Common Problems

Unsupported Graphics Driver Error

This means your graphics card and driver do not have the minimum required OpenGL 3.3 version needed by Blender.

Installing the latest driver can help upgrade the OpenGL version, though some graphics cards are simply too old to run the latest Blender. Using Blender 2.79 or earlier is the only option then.

Crash on Startup

Try running Blender from the command line, to see if any helpful error messages are printed.

On Windows, graphics drivers can sometimes get corrupted. In this case it can help to uninstall all graphics drivers (there may be multiple from Intel, AMD and Nvidia) and perform a clean installation with drivers from the manufacturer's website.

Poor Performance

- Update your graphics drivers (see above).
- On laptops, make sure you are using a dedicated GPU (see above).
- Try lowering quality settings in *Preferences* ▸ *System* ▸ *Memory & Limits*.
- Try undoing settings in your graphics drivers, if you made any changes there.

Render Errors

See EEVEE and Cycles documentation respectively.

Wrong Selection in 3D Viewport

See Invalid Selection, Disable Anti-Aliasing.

Virtual Machines

Running Blender inside a virtual machine is known to have problems when OpenGL drawing calls are forwarded to the host operating system.

To resolve this, configure the system to use PCI passthrough.

Information

To find out which graphics card and driver Blender is using, use *Help ▸ Save System Info* inside Blender. The OpenGL section will have information about your graphics card, vendor and driver version.